计概Cheat Sheet

Edited 20251221 1156

0 语法

0.1 输入输出

不定行输入的几种读法：

```
#try...except捕获异常
while True:
    try:
        #正常读取
    except EOFError:
        break
```

利用sys逐行读取

```
import sys
for line in sys.stdin:
    line = line.strip()
    #处理每一行

#同样适用于一般的输入读取特别是大量数据输入流优化
data = sys.stdin.read()
lines = data.splitlines()
for line in lines:
    #处理每一行
```

it、next指针

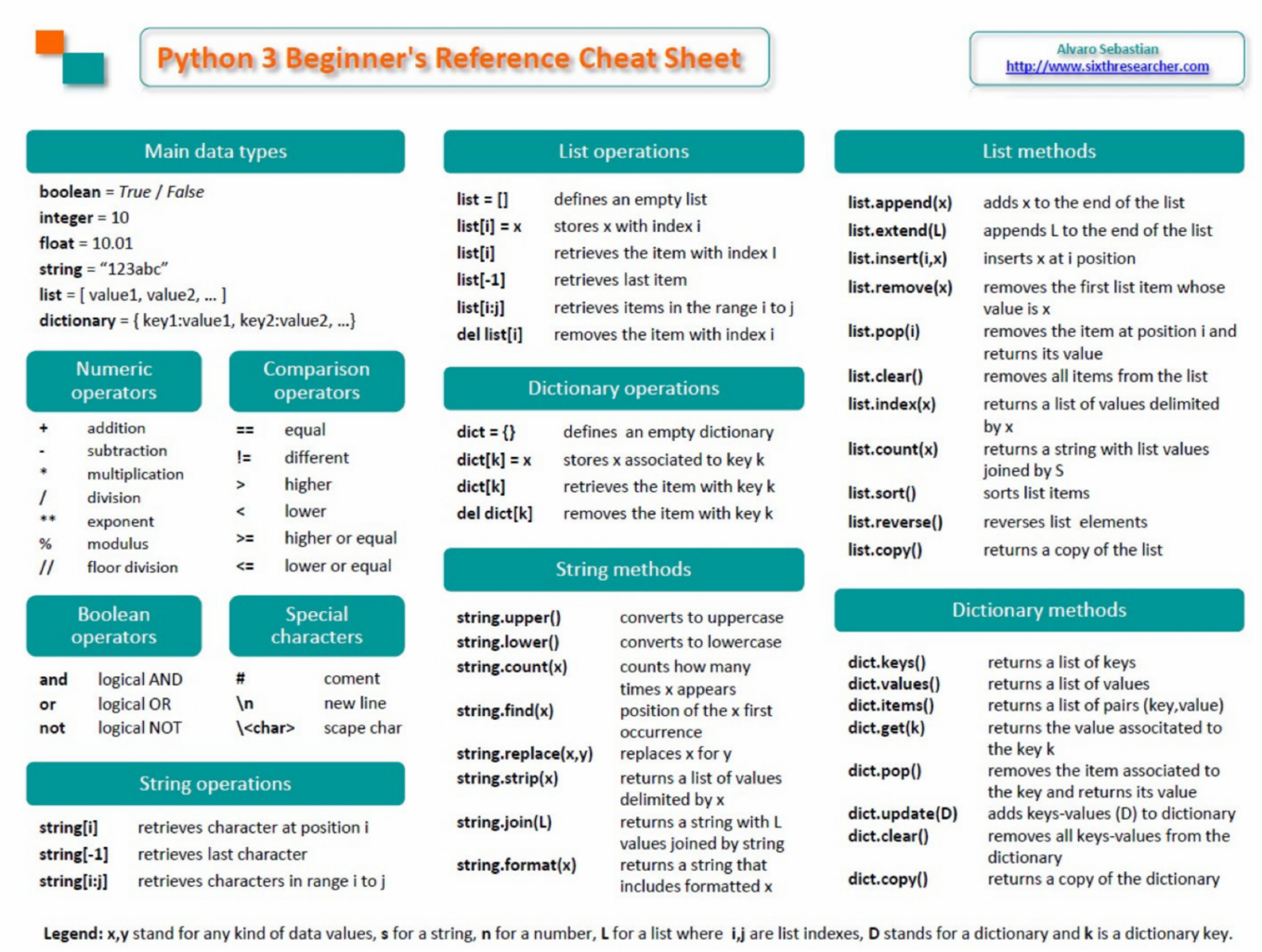
```
import sys
data = sys.stdin.read().split()
it = iter(data)
n = int(next(it))
```

0.2 可变与不可变对象

常用不可变对象：整数，浮点数，字符串，元组。

常用可变对象：列表，字典，集合。

==重要：==可变对象作为参数传入函数，若在函数内被修改，会影响外部变量。



Python语言基本数据类型操作

0.3 defaultdict

工厂函数（factory function）是什么？ 在 defaultdict(factory) 里，factory 就是一个可以被调用的函数，用来生成默认值。 如果 key 不存在，defaultdict 就会调用这个函数创建一个新对象。 常见工厂函数： list → 默认值是空列表 [] set → 默认值是空集合 set() int → 默认值是 0（常用于计数器） 例子：

```
from collections import defaultdict
dd_list = defaultdict(list)   # 工厂函数是 list
dd_set  = defaultdict(set)    # 工厂函数是 set
dd_int  = defaultdict(int)    # 工厂函数是 int
print(dd_list['x'])  # []
print(dd_set['y'])   # set()
print(dd_int['z'])   # 0
```

这里的 list, set, int 就是"工厂函数"：它们不需要参数，就能返回一个新对象。

0.4 OOP写法

例：并查集 Union find

```
class UnionFind:
    def __init__(self,n):
        self.parent=list(range(n))
        self.rank=[0]*n
    def find(self, x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]
    def union(self,x,y):#按秩合并
        rx,ry = self.find(x),self.find(y)
        if rx == ry:
            return False
        if self.rank[rx] < self.rank[ry]:
            self.parent[rx] = ry
        elif self.rank[rx] > self.rank[ry]:
            self.parent[ry] = rx
        else:
            self.parent[ry] = rx
            self.rank[rx]+=1
        return True
# input: n, m
uf = UnionFind(n+1)
for _ in range(m):
    p,q = map(int,input().split())
    uf.union(p,q)
#注意！计数需要先使用find，确保同一类的元素的parent为同一代表元
myset = {uf.find(x) for x in range(1,n+1)}
print(f"Case {case}: {len(myset)}")
```

1. oj有自定义函数时出现奇怪的compile error

在代码第一行加

```
# pylint: skip-file
```

2. bfs模板

示例：sy319矩阵中的块

https://sunnywhy.com/sfbj/8/2/319

题目描述

现有一个 n*m 的矩阵，矩阵中的元素为0或1。然后进行如下定义：

1. 位置(x,y)与其上下左右四个位置 $(x,y + 1)、(x,y - 1)、(x + 1,y)、(x-1,y)$ 是相邻的；
2. 如果位置 (x1,y1) 与位置 (x2,y2) 相邻，且位置 (x2,y2) 与位置 (x3,y3) 相邻，那么称位置(x1,y1)与位置(x3,y3)也相邻；

3. 称个数尽可能多的相邻的1构成一个“块”。

求给定的矩阵中“块”的个数。

输入

第一行两个整数 n、m（$2 \le n \le 100, 2 \le m \le 100$），分别表示矩阵的行数和列数；

接下来 n 行，每行 m 个0或1（用空格隔开），表示矩阵中的所有元素。

输出

输出一个整数，表示矩阵中“块”的个数。

加保护圈，inq_set集合判断是否入过队

```
from collections import deque

# Constants
MAXD = 4
dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

def bfs(x, y):
    q = deque([(x, y)])
    inq_set.add((x,y))
    while q:
        front = q.popleft()
        for i in range(MAXD):
            next_x = front[0] + dx[i]
            next_y = front[1] + dy[i]
            if matrix[next_x][next_y] == 1 and (next_x,next_y) not in inq_set:
                inq_set.add((next_x, next_y))
                q.append((next_x, next_y))

# Input
n, m = map(int, input().split())
matrix=[[-1]*(m+2)]+[[-1]+list(map(int,input().split()))+[-1] for i in range(n)]+
[[-1]*(m+2)]
inq_set = set()

# Main process
counter = 0
for i in range(1,n+1):
    for j in range(1,m+1):
        if matrix[i][j] == 1 and (i,j) not in inq_set:
            bfs(i, j)
            counter += 1

# Output
print(counter)
```

inq 数组，结点是否已入过队

```
from collections import deque

# Constants
MAXN = 100
MAXD = 4
dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

# Functions
def can_visit(x, y):
    return 0 <= x < n and 0 <= y < m and matrix[x][y] == 1 and not in_queue[x][y]

def bfs(x, y):
    q = deque([(x, y)])
    in_queue[x][y] = True
    while q:
        front = q.popleft()
        for i in range(MAXD):
            next_x = front[0] + dx[i]
            next_y = front[1] + dy[i]
            if can_visit(next_x, next_y):
                in_queue[next_x][next_y] = True
                q.append((next_x, next_y))

# Input
n, m = map(int, input().split())
matrix = [list(map(int, input().split())) for _ in range(n)]
in_queue = [[False] * MAXN for _ in range(MAXN)]

# Main process
counter = 0
for i in range(n):
    for j in range(m):
        if matrix[i][j] == 1 and not in_queue[i][j]:
            bfs(i, j)
            counter += 1

# Output
print(counter)
```

变形：sy321迷宫最短路径

https://sunnywhy.com/sfbj/8/2/321

现有一个 n*m 大小的迷宫，其中1表示不可通过的墙壁，0表示平地。每次移动只能向上下左右移动一格，且只能移动到平地上。假设左上角坐标是（1，1），行数增加的方向为x增长的方向，列数增加的方向为y增长的方向，求从迷宫左上角到右下角的最少步数的路径。

思路：用pre[x][y]记录到达该位置的前一个位置，从原点开始搜索，到达右下角停止，最后由终点回溯得到路径。

```
from collections import deque

dx = [0, 1, 0, -1]
dy = [1, 0, -1, 0]

def bfs():
    q = deque([(0, 0)])
    pre[0][0] = (0, 0)

    while q:
        node = q.popleft()
        x = node[0]
        y = node[1]
        for t in range(4):
            nx, ny = x + dx[t], y + dy[t]
            if 0 <= nx < n and 0 <= ny < m and a[nx][ny] == 0 and pre[nx][ny] ==
(-1, -1):
                pre[nx][ny] = (x, y)
                q.append((nx, ny))
                if nx == n - 1 and ny == m - 1:
                    return

n, m = map(int, input().split())
a = [list(map(int, input().split())) for _ in range(n)]
pre = [[(-1, -1)] * m for _ in range(n)]

bfs()
ans = [(n, m)]
x, y = n - 1, m - 1
while x or y:
    x, y = pre[x][y][0], pre[x][y][1]
    ans.append((x + 1, y + 1))
for i in ans[::-1]:
    print(*i)
```

3. dfs模板

示例：sy314指定步数的迷宫问题 中等

https://sunnywhy.com/sfbj/8/1/314

现有一个 n*m 大小的迷宫，其中1表示不可通过的墙壁，0表示平地。每次移动只能向上下左右移动一格（不允许移动到曾经经过的位置），且只能移动到平地上。现从迷宫左上角出发，问能否在恰好第步时到达右下角。

输入

第一行三个整数$n、m、k \hspace{1em} (2 \le n \le5, 2 \le m \le 5, 2 \le k \le n*m)$，分别表示迷宫的行数、列数、移动的步数；

接下来行，每行个整数（值为0或1），表示迷宫。

输出

如果可行，那么输出Yes，否则输出No。

加保护圈，原地修改

```
dx = [-1, 0, 1, 0]
dy = [ 0, 1, 0, -1]

canReach = False
def dfs(maze, x, y, step):
    global canReach
    if canReach:
        return

    for i in range(4):
        nx = x + dx[i]
        ny = y + dy[i]
        if maze[nx][ny] == 'e':
            if step==k-1:
                canReach = True
                return

            continue

        if maze[nx][ny] == 0:
            if step < k:
                maze[x][y] = -1
                dfs(maze, nx, ny, step+1)
                maze[x][y] = 0


n, m, k = map(int, input().split())
maze = []
maze.append( [-1 for x in range(m+2)] )
for _ in range(n):
    maze.append([-1] + [int(_) for _ in input().split()] + [-1])
maze.append( [-1 for x in range(m+2)] )

maze[1][1] = 's'
maze[n][m] = 'e'

dfs(maze, 1, 1, 0)
print("Yes" if canReach else "No")
```

辅助visited空间

```
MAXN = 5
n, m, k = map(int, input().split())
maze = []
for _ in range(n):
    row = list(map(int, input().split()))
    maze.append(row)

visited = [[False for _ in range(m)] for _ in range(n)]
canReach = False

MAXD = 4
dx = [0, 0, 1, -1]
dy = [1, -1, 0, 0]

def is_valid(x, y):
    return 0 <= x < n and 0 <= y < m and maze[x][y] == 0 and not visited[x][y]

def DFS(x, y, step):
    global canReach
    if canReach:
        return
    if x == n - 1 and y == m - 1:
        if step == k:
            canReach = True
        return
    visited[x][y] = True
    for i in range(MAXD):
        nextX = x + dx[i]
        nextY = y + dy[i]
        if step < k and is_valid(nextX, nextY):
            DFS(nextX, nextY, step + 1)
    visited[x][y] = False

DFS(0, 0, 0)
print("Yes" if canReach else "No")
```

小技巧：

递归爆栈可考虑调整递归深度限制：

```
import sys
sys.setrecursionlimit(1<<30)
```

lru_cache 记忆化搜索

```
# 数的划分
from functools import lru_cache
```

```
@lru_cache(maxsize = None)
def dfs(n, k, x):
    if k == 1:
        return 1
    ans = 0
    for i in range((n + k - 1) // k, min(n - k + 1, x) + 1):
        ans += dfs(n - i, k - 1, i)
    return ans
```

4. heapq

```
    Usage:

    heap = []            # creates an empty heap
    heappush(heap, item) # pushes a new item on the heap
    item = heappop(heap) # pops the smallest item from the heap
    item = heap[0]       # smallest item on the heap without popping it
    heapify(x)           # transforms list into a heap, in-place, in linear time
    item = heappushpop(heap, item) # pushes a new item and then returns
                                   # the smallest item; the heap size is unchanged
    item = heapreplace(heap, item) # pops and returns smallest item, and adds
                                   # new item; the heap size is unchanged
```

5. 欧拉筛

O(n)

```
import math
n=int(input())
primes=[]
#numbers=[True for i in range(int(1e6+1))]
numbers=[True]*(10**6+1)
numbers[0]=False
numbers[1]=False
def sieve(numbers):
    for i in range(2, int(1e6+1)):
        if numbers[i]:
            primes.append(i)
        for j in range(len(primes)):
            if i*primes[j]>int(1e6):
                break
            numbers[i*primes[j]]=False
            if i%primes[j]==0:
                break
sieve(numbers)
```

6. Dijkstra

Dijkstra 用来处理图中的最短路径问题

例：走山路

```
from heapq import heappush, heappop
dx = [1, 0, -1, 0]
dy = [0, 1, 0, -1]
inf = float('inf')
def dijkstra(sx, sy, tx, ty):
    if h[sx][sy] == '#' or h[tx][ty] == '#': return inf
    q = []
    d = [[inf] * n for _ in range(m)]
    d[sx][sy] = 0
    heappush(q, (0, sx, sy))
    while q:
        dist, ux, uy = heappop(q)
        if d[ux][uy] != dist: continue
        if (ux, uy) == (tx, ty): return dist
        for t in range(4):
            nx, ny = ux + dx[t], uy + dy[t]
            if nx < 0 or nx >= m or ny < 0 or ny >= n: continue
            if h[nx][ny] == '#': continue
            now_dist = dist + abs(int(h[ux][uy]) - int(h[nx][ny]))
            if now_dist < d[nx][ny]:
                d[nx][ny] = now_dist
                heappush(q, (now_dist, nx, ny))
    return d[tx][ty]
m, n, p = map(int, input().split())
h = []
for _ in range(m):
    h.append(input().split())
for _ in range(p):
    sx, sy, tx, ty = map(int, input().split())
    min_d = dijkstra(sx, sy, tx, ty)
    if min_d == inf: print('NO')
    else: print(min_d)
```

7. dp

dp

最长公共子序列

```
#子序列不一定连续
a=list(input())
b=list(input())
dp=[[0 for i in range(len(b))] for j in range(len(a))]
for i in range(len(a)):
    for j in range(len(b)):
        if a[i] == b[j]:
```

```
            dp[i][j] = dp[i-1][j-1]+1
        else:
            dp[i][j] = max(dp[i-1][j],dp[i][j-1])
print(dp[-1][-1])
```

最长单调增子序列

```
a=list(map(int, input().split()))
n=len(a)
dp = [1]*n
for i in range(1,n):
    for j in range(i):
        if a[j]<a[i]:
            dp[i] = max(dp[i],dp[j]+1)
ans = max(dp)
print(ans)
```

利用bisect优化 O(nlogn)

```
import bisect
n = int(input())
*lis, = map(int, input().split())
dp = [1e9]*n
for i in lis:
    dp[bisect.bisect_left(dp, i)] = i
print(bisect.bisect_left(dp, 1e8))
```

背包问题

0-1背包

例：小偷背包。在满足背包容量的情况下，使偷的价值最高

```
dp = [0] + [-1] * b
for i in range(n):
    for j in range(b - w[i], -1, -1):
        if dp[j] != -1:
            dp[j + w[i]] = max(dp[j + w[i]], dp[j] + val[i])
print(max(dp))
```

这里采用"**滚动数组**"的方法将二维数组压缩成一维，是DP问题中常用的技巧。这基于选前i个物品的状态仅依赖于选前i-1个物品的状态。注意**内层循环要倒着遍历**！

完全背包

将0-1背包中内层循环改为正着遍历即可（这里其实就利用了**先前已经得到的信息**来简化转移：在先前的转移中物品i可能已经用过若干次了）

多重背包

每个物品的个数有限制,把每个物品的个数拆成1 2 4等转化为01背包

```
#NBA门票
def many_bag():
    #s=[0, ...  ]为每个物品的个数
    dp=[0]*(V+1)
    for i in range(1, n + 1):
        k=1
        while s[i] > 0:
            cnt = min(k, s[i])
            for j in range(V, cnt * cost[i] - 1, -1):
                dp[j]=max(dp[j], cnt * price[i] + dp[j - cnt * cost[i]])
            s[i] -= cnt
            k *= 2
    return dp[-1]
```

更换状态，记录空余的物品数量。例：Coins

设 F[i][j] 表示使用前 i 种物品填满容量 j 的背包后，最多剩余的第 i 种物品数量。若 F[i][j]≥0 表示状态可行

```
    dp = [0] + [-1] * m
    for i in range(n):
        for j in range(m + 1):
            if dp[j] >= 0: dp[j] = c[i]
        for j in range(m - a[i] + 1):
            if dp[j] > 0:
                dp[j + a[i]] = max(dp[j + a[i]], dp[j] - 1)
    ans = 0
    for i in range(1, m + 1):
        if dp[i] >= 0: ans += 1
    print(ans)
```

注：背包问题的DP解法需要重量M不太大，因为要遍历每个可能的M。如果M很大而物品数量很少，采用DFS枚举物品的选法有时是更好的选择。

8. 二分查找

```
# 河中跳房子
l = 1
r = L + 1
while r - l > 1:
    mid = (l + r) // 2
    if stone(mid) <= m : l = mid
```

```
        else: r = mid
    print(l)
```

bisect库

python内置二分查找工具

注意bisect库只适用于**升序序列**，对于降序序列需要将列中每个数取相反数再使用；如果要按指定的key排序，可以将每个元素变为元组，把key放到元组的第一个形成元组序列

bisect_left和bisect_right分别表示插入可能位置中最靠左/右的位置；注意返回的是下标。

```
bisect_left(a, x, lo=0, hi=None, *, key=None)
    Return the index where to insert item x in list a, assuming a is sorted.

    The return value i is such that all e in a[:i] have e < x, and all e in a[i:]
have e >= x.  So if x already appears in the list, a.insert(i, x) will insert just
**before** the leftmost x already there.

    Optional args lo (default 0) and hi (default len(a)) bound the
    slice of a to be searched.

bisect_right(a, x, lo=0, hi=None, *, key=None)
    Return the index where to insert item x in list a, assuming a is sorted.

    The return value i is such that all e in a[:i] have e <= x, and all e in a[i:]
have e > x.  So if x already appears in the list, a.insert(i, x) will insert just
after the rightmost x already there.
```

应用

一是方程求解问题。注意这个时候while循环的退出条件应当是l和r的差小于所需精度。

二是一类最优化问题，特别是“最值的最值”问题。这类问题所求的最优值通常具有“单调性质”，即小于某个数的都可以但大于它的都不行。对于这种问题，可以考虑**直接去枚举**最优值的可能取值。利用单调性质采用二分算法，这一步只有logn的复杂度；接下来的问题转化为**判断该最优值**是否满足要求。这样就把最优化问题转化为了判定问题，而判定问题有时有比较好的办法解决。所谓“0-1分数规划问题”也能用类似方法解决。