**Python Cheat Sheet - SOI Workshop Zürich 2017**

**Recommended PyCharm Setup**
To submit, remove sys.stdin=open("sample01.in").

**Python Docs**
https://docs.python.org/

**I/O**
string = input() # read line as string
number = int(input()) # read line as integer
print(string) # prints string with newline
print(number, string) # print separated by space
print("Impossible") # print a string literal
print(f"Case #{testcase}: {solution}"") # formatting

**Files**
import sys
sys.stdin = open("sample01.in", "r")
Remove those lines before submitting at grader.soi.ch!

**Loops**
for i in range(n): # for loop
    print(i) # prints 0, 1, 2, 3, 4
for i in range(2, 14, 4): # start, stop, step
    print(i) # prints 2, 6, 10
while loops repeat as long as the condition remains true:
    i = 5
while i > 0: # while loop
    i -= 1 # i has values 5, 4, 3, 2, 1

**Tuples**
tup = 3, 4, 5 # create tuple of two elements
a, b, c = tup # extract tuple: a=3, b=4, c=5
_, _, c = tup # when you don't care about some elements, use _
c = tup[2] # indexing works

**Lists (can be used as stacks)**
Basics:
numbers = [] # create new list
for i in range(5):
    numbers.append(int(input())) # add new element
for x in numbers: # iterate over list
    print(x)
Operations:
numbers.append(3) # O(1) append at the end
x = numbers.pop() # O(1) remove last element
numbers = [1] + numbers # O(len(numbers))
Indexing:
• a[0]: first element
• a[len(a) - 1]: last element
• a[-1]: last element
• a[-len(a)]: first element
Slices:
• a[3:6]: elements at indices 3, 4, 5
• a[1:3]: elements at indices 0, 1, 2
• a[3:]: elements at indices 3, 4, ..., len(a)-1
• a[:]: copy of whole list
List comprehensions:
numbers = [3, 1, 4, 1, 5, 9] # create new list
doubled = [2*x for x in numbers] # list comprehension
Useful functions:
sum(numbers) # sum of all elements
max(numbers) # maximum of all elements
min(numbers) # minimum of all elements
max(numbers, key=lambda x: x%7) # maximum of all elements
sorted(numbers, key=lambda x: x%7) # maximum of all elements

**Sets**
Like a dict, but only stores keys. Elements are stored only once.
s = set() # create an empty set
s = {1, 2} # prefilled
s = set(values) # convert list to set
# WARNING: s = {} creates a dict, not a set
s.add(1) # O(1) add element to set
for x in s: # O(1) check if element exists
    s.remove(2) # O(1) remove element
for element in s: # O(len(s)) iterate
    print(s) # careful: not sorted

**Deques (can be used as queues)**
from collections import deque
def (q) # empty
q.append(4) # O(1) append at the end
q.appendleft(4) # O(1) append at the front
x = q.pop() # O(1) remove last element
y = q.popleft() # O(1) remove first element

**Min-Heaps/Priority Queues**
Datastructure with fast insert and fast access to the minimum. (Trick: to sort it in reserve, insert -x.)

**Dictionaries**
Stores key-value pairs with fast key lookup. All keys are unique.
m = {} # create an empty dict
m = {"Turing": 1954: "Newton": 1727} # prefilled
m = {x: 2**x for x in range(10)} # list compr.
m["Einstein"] # O(1) insert or modify
print("Einstein" in m) # O(1) check if key exists
del m["Einstein"] # O(1) remove item
for key, value in m.items(): # O(len(m)) iterate
    print(f"key: (key) -> (value)")

**Using map to parse and print lists**
map(f, values) behaves like [f(x) for x in values], except that it returns a generator and not a list.
n, m = map(int, input().split()) # extract directly
numbers = list(map(int, input().split())) # to store many numbers, convert them to a list
numbers = list(map(int, input().split())) # map
numbers = list(map(int, input().split())) # map

**Lambdas**
negate = lambda x: -x # negate(3) == -3
add = lambda a, b: a + b # add(1, 3) == 4
# use lambdas as key functions
sorted(numbers, key=lambda x: x%7)