

### 3. بـ ترتیب

```
input_string = input("Enter a string: ")
```

```
letters = []
```

```
digits = []
```

```
others = []
```

```
for char in input_string:
```

```
    if char.isalpha():
```

```
        letters.append(char)
```

```
    elif char.isdigit():
```

```
        digits.append(char)
```

```
    else:
```

```
        others.append(char)
```

```
letters.sort(key=lambda x: (x.lower(),
```

```
x.isupper()))
```

```
digits.sort()
```

```
others.sort()
```

```
result = ".join(others) + ".join(digits) +
```

```
".join(letters)
```

```
print("Sorted output:", result)
```

در ادامه کد را بدون استفاده از تابع و به صورت ساده ارائه می‌دهم:

```
# دریافت ورودی از کاربر
input_string = input("")
```

# لیست برای دسته‌بندی انواع کاراکترها

```
letters = []
digits = []
others = []
```

# دسته‌بندی کاراکترها

```
for char in input_string:
```

اگر حرف است #

```
    letters.append(char)
```

اگر عدد است #

```
    digits.append(char)
```

```
    digits.append(char)
else: # اگر کاراکتر خاص است
    others.append(char)
```

```
# مرتبسازی هر دسته
letters.sort(key=lambda x: (x.lower(),
x.isupper())) # مرتبسازی حروف (بر اساس کوچک/بزرگ)
digits.sort() # مرتبسازی اعداد
others.sort() # مرتبسازی کاراکترهای خاص
```

```
# ترکیب نتیجه به ترتیب: اعداد + حروف + سایر
result = ".join(others) + ".join(digits) +
".join(letters)
```

```
# نمایش خروجی
print("خروجی مرتب شده:", result)
```

---

## توضیحات:

1. ابتدا ورودی از کاربر دریافت می‌شود.

2. سه لیست برای دسته‌بندی حروف، اعداد و سایر کاراکترها تعریف می‌شود.

3. با یک حلقه for تمام کاراکترهای ورودی بررسی و به دسته‌های مناسب اضافه می‌شوند:

برای حروف letters

برای اعداد digits

برای کاراکترهای خاص others

4. هر لیست به صورت جداگانه مرتب می‌شود:

حروف (با در نظر گرفتن کوچک/بزرگ)

اعداد به ترتیب عددی

کاراکترهای خاص

5. لیست‌ها در ترتیب خواسته شده به هم متصل می‌شوند و نتیجه به کاربر نمایش داده می‌شود.

---

تست نمونه:

ورودی:

Yek CCharactere kheYli AJ!!B ghA@\*Rib  
Mine#\$#visim

خروجى:

!!@#\$#\*AaABCceghiiikkklmnrrstvy

# مشكل ايمان 1

```
grades = {'A': 5, 'B': 4, 'C': 3, 'D': 2, 'E': 1, 'F': 0}
```

```
n = int(input())
```

```
results = []
```

```
for _ in range(n):
```

```
    line = input().split()
```

```
    weight = int(line[0])
```

```
    grade = line[1]
```

```
    results.append((grades[grade], weight,  
grade))
```

```
results.sort(reverse=True, key=lambda x:  
(x[0], x[1]))
```

```
for _, weight, grade in results:
```

```
    print(weight, grade)
```

برای حل این تمرین کد زیر می‌تواند مورد استفاده قرار گیرد. این کد ورودی را دریافت کرده، وزن و تعداد نمرات را پردازش کرده و خروجی مورد نظر را تولید می‌کند.

# تعریف مقدار نمرات

```
grades = {'A': 5, 'B': 4, 'C': 3, 'D': 2, 'E': 1, 'F': 0}
```

# دریافت ورودی

```
n = int(input()) # تعداد امتحانات
```

```
results = []
```

```
for _ in range(n):
```

```
    line = input().split()
```

وزن امتحان #

```
grade = line[1] # نمره امتحان
```

```
results.append((grades[grade], weight,
```

```
grade))
```

# مرتبسازی بر اساس نمره و سپس وزن

```
results.sort(reverse=True, key=lambda x:  
(x[0], x[1]))
```

```
# چاپ نتایج  
for _, weight, grade in results:  
    print(weight, grade)
```

توضیح کد:

1. دیکشنری نمرات: مقادیر مربوط به هر نمره (A تا F) مشخص می‌شوند.

2. دریافت ورودی: ابتدا تعداد امتحانات ( $n$ ) دریافت می‌شود. سپس در حلقه، وزن و نمره هر امتحان به صورت جداگانه گرفته شده و پردازش می‌شود.

3. ایجاد لیست نتایج: یک لیست از سهتایی‌ها (نمره، وزن، نمره اصلی) ایجاد می‌شود.

4. مرتبسازی: بر اساس نمره و در صورت برابر بودن، وزن به ترتیب نزولی مرتب می‌شود.

5. چاپ: نتایج مرتب شده چاپ می‌شوند.

ورودی نمونه:

4

3 D

2 A

3 D

1 B

خروجی نمونه:

2 A

1 B

3 D

3 D

## 2. خطر در جو جو

```
n = int(input())
```

```
m = int(input())
```

```
count_removed = 0
```

```
result = -1
```

```
for i in range(2, n + 1):
```

```
    is_prime = True
```

```
    for j in range(2, int(i**0.5) + 1):
```

```
        if i % j == 0:
```

```
            is_prime = False
```

```
            break
```

```
if is_prime:
```

```
    continue
```

```
for k in range(i, n + 1, i):
```

```
    count_removed += 1
```

```
    if count_removed == m:
```

```
result = i
```

```
if result == -1:
```

```
    print(-1)
```

```
else:
```

```
    print(result)
```

در اینجا کد به صورت گام با توضیحات هر بخش توضیح داده شده است:

---

کد کامل با توضیحات:

n = int(input()) # مقدار n بزرگ‌ترین عدد در غربال است

m = int(input()) # مقدار m شماره عدد حذف شده است

شمارنده تعداد اعداد #

حذف شده

اول باشد، m خروجی نهایی، اگر عدد #

مقدار -1 باقی می‌ماند

# اجرای غربال اراتستن بدون استفاده از لیست

n شروع از عدد 2 تا #

اول i فرض می‌کنیم عدد #

است

# بررسی اول بودن عدد i

بررسی #

i مقسوم‌علیه‌ها از 2 تا ریشه مربع

بخش‌پذیر باشد زیرا اگر عدد #

اول نیست | عدد | عدد #

از حلقه خارج می‌شویم #

if is\_prime:

اگر عدد اول باشد، خط نمی‌خورد #

و به عدد بعدی می‌رویم

# حذف مضارب عدد i

for k in range(i, n + 1, i): # حلقه‌ای برای

ز حذف تمام مضارب

هر بار که عددی count\_removed += 1 #

حذف شود، شمارنده افزایش می‌یابد

if count\_removed == m: # اگر به عدد m

رسیدیم

مقدار عددی که باعث حذف i #

شد، ذخیره می‌کنیم

# بررسی خروجی

if result == -1:

اول بود، مقدار -1 چاپ # اگر عدد m

می‌شود

else:

در غیر این صورت، عددی که print(result) #

امین عدد حذف شده است چاپ می‌شود

---

## توضیح گام به گام:

### 1. ورودی گرفتن:

مقدار  $n$  به عنوان بزرگ‌ترین عدد غربال دریافت می‌شود.

مقدار  $m$  به عنوان شماره عددی که باید حذف شود گرفته می‌شود.

### 2. متغیرهای اولیه:

تعداد اعداد حذف شده را دنبال می‌کند.

عددی که باید در نتیجه چاپ شود. اگر عدد  $m$  باشد، مقدار آن - 1 باقی می‌ماند.

### 3. حلقه اصلی (برای هر عدد از 2 تا n):

هر عدد را بررسی می‌شود.

ابتدا فرض می‌کنیم عدد را اول است (True).

سپس با استفاده از حلقه‌ای از 2 تا ریشه مربع عدد را بررسی می‌کنیم که آیا عدد را بر مقسم علیه هایش بخش‌پذیر است یا خیر.

اگر عدد بخش‌پذیر باشد، is\_prime = False تنظیم می‌شود، و از بررسی بیشتر برای این عدد صرف نظر می‌کنیم.

### 4. اعداد اول:

اگر عدد  $i$  اول باشد، حذف نمی‌شود و به عدد بعدی می‌رویم (continue).

## 5. حذف مضارب:

اگر عدد  $i$  اول نبود، وارد حلقه‌ای می‌شویم که تمامی مضارب عدد  $i$  را از  $\pi$  تا  $n$  حذف می‌کند.

به ازای هر عدد حذف شده، شمارنده یک واحد افزایش می‌یابد.

اگر شماره حذف شده با  $m$  برابر شود، عدد  $i$  به عنوان نتیجه ثبت می‌شود.

## 6. بررسی خروجی:

اگر هیچ عددی در جایگاه  $m$  حذف نشود، مقدار

اول  $m$  برابر ۱ باقی می‌ماند، که نشان می‌دهد است.

در غیر این صورت، عددی که امین عدد حذف شده است به عنوان نتیجه چاپ می‌شود.

---

مثالی از اجرا:

ورودی:

5  
1

خروجی:

توضیح:

1. غربال اراتستن برای اعداد تا 5 اجرا می‌شود.
2. اعداد 4 و 6 به ترتیب به عنوان اولین و دومین عدد حذف شده انتخاب می‌شوند.
3. در این مثال، اولین عدد حذف شده 4 است، بنابراین نتیجه برابر با 4 است.

`n = int(input()) # مقدار n`  
که بزرگ‌ترین عدد در n مقدار است  
غربال است

`m = int(input()) # مقدار m`  
که شماره عدد m مقدار است  
حذف شده است

`count_removed = 0 # شمارنده تعداد اعداد حذف شده`

اول باشد، m خروجی نهايی، اگر عدد # -1 # خروجی نهايی، اگر عدد # -1  
مقدار -1 باقی می‌ماند

# اجرای غربال اراتستن بدون استفاده از لیست  
for i in range(2, n + 1): # شروع از عدد 2 تا n  
اول | فرض می‌کنیم عدد # کنیم عدد #  
است

# بررسی اول بودن عدد i  
بررسی # for j in range(2, int(i\*\*0.5) + 1): #  
| مقسوم‌علیه‌ها از 2 تا ریشه مربع  
بخش‌پذیر باشد زیرا اگر عدد #  
اول نیست | عدد #  
از حلقه خارج می‌شویم # break #

if is\_prime:  
اگر عدد اول باشد، خط نمی‌خورد #  
و به عدد بعدی می‌رویم continue #

# حذف مضارب عدد i  
حلقه‌ای برای # for k in range(i, n + 1, i): #

حذف تمام مضارب i

هر بار که عددی  $count\_removed += 1$  #

حذف شود، شمارنده افزایش می‌یابد

اگر به عدد m if  $count\_removed == m$ : #

رسیدیم

مقدار عددی که باعث حذف i #

شد، ذخیره می‌کنیم

# بررسی خروجی

if result == -1:

اول بود، مقدار -1 چاپ m اگر عدد #

می‌شود

else:

در غیر این صورت، عددی که print(result) #

امین عدد حذف شده است چاپ می‌شود

## 2. دور دور هوایی

```
n, m = map(int, input().split())
cities = [input().strip() for _ in range(n)]
flights = [input().strip().split() for _ in
range(m)]
```

```
connections = {city: [] for city in cities}
```

```
for source, destination in flights:
    connections[source].append(destination)
```

```
for city in cities:
    direct_flights = connections[city]
    print(len(direct_flights))
    if direct_flights:
        print("\n".join(direct_flights))
```

# دریافت تعداد شهرها (n) و تعداد خطوط هوایی (m)

```
n, m = map(int, input().split())
```

# دریافت نام شهرها و ذخیره آنها در لیست

```
cities = [input().strip() for _ in range(n)]
```

# دریافت خطوط هوایی (مبدأ و مقصد هر پرواز) و ذخیره در لیست flights

```
flights = [input().strip().split() for _ in range(m)]
```

# ایجاد یک دیکشنری برای ذخیره پروازهای مستقیم

هر شهر

# کلیدها نام شهرها هستند و مقدارها لیستی از شهرهای مقصد

```
connections = {city: [] for city in cities}
```

# پر کردن دیکشنری با اطلاعات خطوط هوایی

for source, destination in flights:

```
connections[source].append(destination)
```

# پردازش و تولید خروجی برای هر شهر

```
for city in cities:
```

# گرفتن لیست پروازهای مستقیم برای شهر جاری

```
direct_flights = connections[city]
```

# چاپ تعداد شهرهایی که به آنها پرواز مستقیم

وجود دارد

```
print(len(direct_flights))
```

# اگر پروازی وجود دارد، نام شهرهای مقصد را در

خطوط جدأگانه چاپ کن

```
if direct_flights:
```

```
    print("\n".join(direct_flights))
```

---

توضیحات کد:

1. ورودی تعداد شهرها و خطوط هوایی:

ابتدا  $n$  (تعداد شهرها) و  $m$  (تعداد خطوط هوایی) از ورودی گرفته می‌شود.

این مقادیر به ترتیب مشخص می‌کنند چند شهر داریم و چند پرواز بین این شهرها وجود دارد.

## 2. دریافت نام شهرها:

با استفاده از یک حلقه،  $n$  خط بعدی به عنوان نام شهرها خوانده شده و در یک لیست به نام `cities` ذخیره می‌شوند.

## 3. دریافت خطوط هوایی:

خطوط هوایی که شامل مبدأ و مقصد هر پرواز هستند، در  $m$  خط بعدی وارد می‌شوند و در قالب جفت (مبدأ، مقصد) ذخیره می‌شوند.

#### 4. ایجاد دیکشنری :connections

این دیکشنری برای ذخیره شهرهایی که هر شهر به آنها پرواز مستقیم دارد، استفاده می‌شود.

کلیدهای دیکشنری نام شهرها هستند و مقدار آنها لیستی است که شهرهای مقصد را شامل می‌شود.

#### 5. پر کردن دیکشنری با خطوط هوایی:

برای هر جفت (مبدأ، مقصد) در لیست flights، شهر مقصد به لیست پروازهای مستقیم مبدأ در دیکشنری اضافه می‌شود.

## 6. تولید خروجی برای هر شهر:

برای هر شهر در لیست :cities

ابتدا تعداد شهرهایی که به آنها پرواز مستقیم دارد با len(direct\_flights) چاپ می‌شود.

سپس اگر پرواز مستقیم وجود دارد، نام شهرهای مقصد در خطوط جداگانه چاپ می‌شود.

---

مثال:

ورودی:

4 3

Tehran

Esfahan

Shiraz

Yazd

Tehran Yazd

Tehran Shiraz

Esfahan Yazd

مراحل اجرا:

n = 4 . تعداد شهرها: 1

m = 3 . تعداد پروازها: 2

3. لیست شهرها: ] 'Tehran', 'Esfahan', 'Shiraz', ]

["Yazd

4. خطوط هوایی:

```
[('Tehran', 'Yazd'), ('Tehran', 'Shiraz'),  
 ('Esfahan', 'Yazd')]
```

5. دیکشنری :connections

```
connections = {  
    "Tehran": ["Yazd", "Shiraz"],  
    "Esfahan": ["Yazd"],  
    "Shiraz": [],  
    "Yazd": []  
}
```

خروجی:

2

Yazd

Shiraz

1

Yazd

0

0

.(Yazd, Shiraz) 2 مقصد دارد

.(Yazd) 1 مقصد دارد

برای شیراز و یزد: هیچ پرواز مستقیمی ندارند.

#### 4. پادشاه ژوزفوس

```
tests = int(input())
```

```
for _ in range(tests):
    line = input().strip()
    if line == "*":
        break
```

```
parts = line.split()
n = int(parts[0]) if len(parts) > 0 else 100
k = int(parts[1]) if len(parts) > 1 else 2
```

```
result = 0
for i in range(2, n + 1):
    result = (result + k) % i
```

```
print(result + 1)
```

توضیحات کد:

## 1. ورودی تعداد آزمایش‌ها:

```
tests = int(input())
```

تعداد دفعاتی که باید آزمایش ژوزفوس انجام شود از ورودی گرفته می‌شود.

## 2. حلقه برای هر آزمایش:

```
for _ in range(tests):
    line = input().strip()
    if line == "*":
        break
```

برای هر آزمایش، یک خط ورودی خوانده می‌شود.

اگر ورودی \* باشد، حلقه متوقف می‌شود.

### 3. تنظیم مقادیر پیشفرض:

```
parts = line.split()
```

```
n = int(parts[0]) if len(parts) > 0 else 100
```

```
k = int(parts[1]) if len(parts) > 1 else 2
```

اگر مقدار  $n$  داده نشده باشد، مقدار پیشفرض 100 در نظر گرفته می‌شود.

اگر مقدار  $k$  داده نشده باشد، مقدار پیشفرض 2 در نظر گرفته می‌شود.

### 4. محاسبه بازمانده ژوزفوس:

```
result = 0
```

```
for i in range(2, n + 1):
```

```
result = (result + k) % i
```

این الگوریتم از رابطه ژوزفوس برای بازمانده در دایره استفاده می‌کند:

در هر گام، بازمانده جدید با توجه به تعداد افراد باقی‌مانده ( $i$ ) محاسبه می‌شود.

حلقه از 2 تا  $n+1$  اجرا می‌شود تا تمام مراحل حذف افراد انجام شود.

5. نمایش نتیجه:

```
print(result + 1)
```

مقدار `result` در الگوریتم 0-مبنای است (شروع از 0).

برای تطبیق با شماره افراد در دایره (1-مبنای)، مقدار 1

به نتیجه اضافه می‌شود.

روزینا نوری