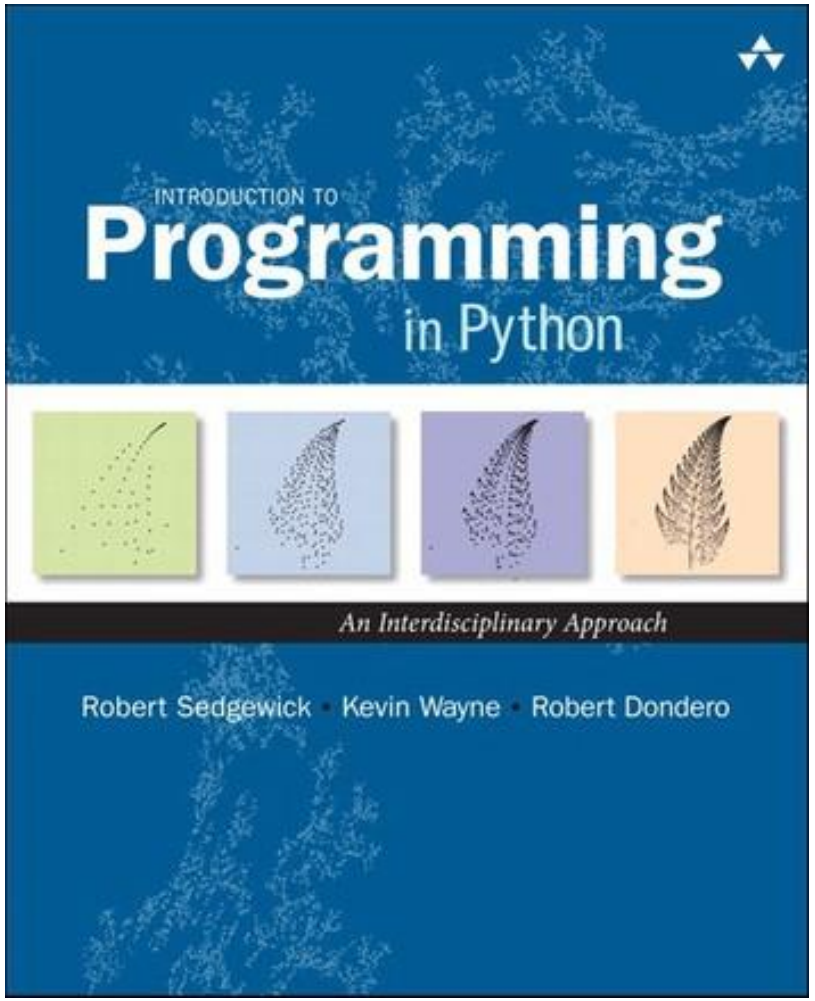
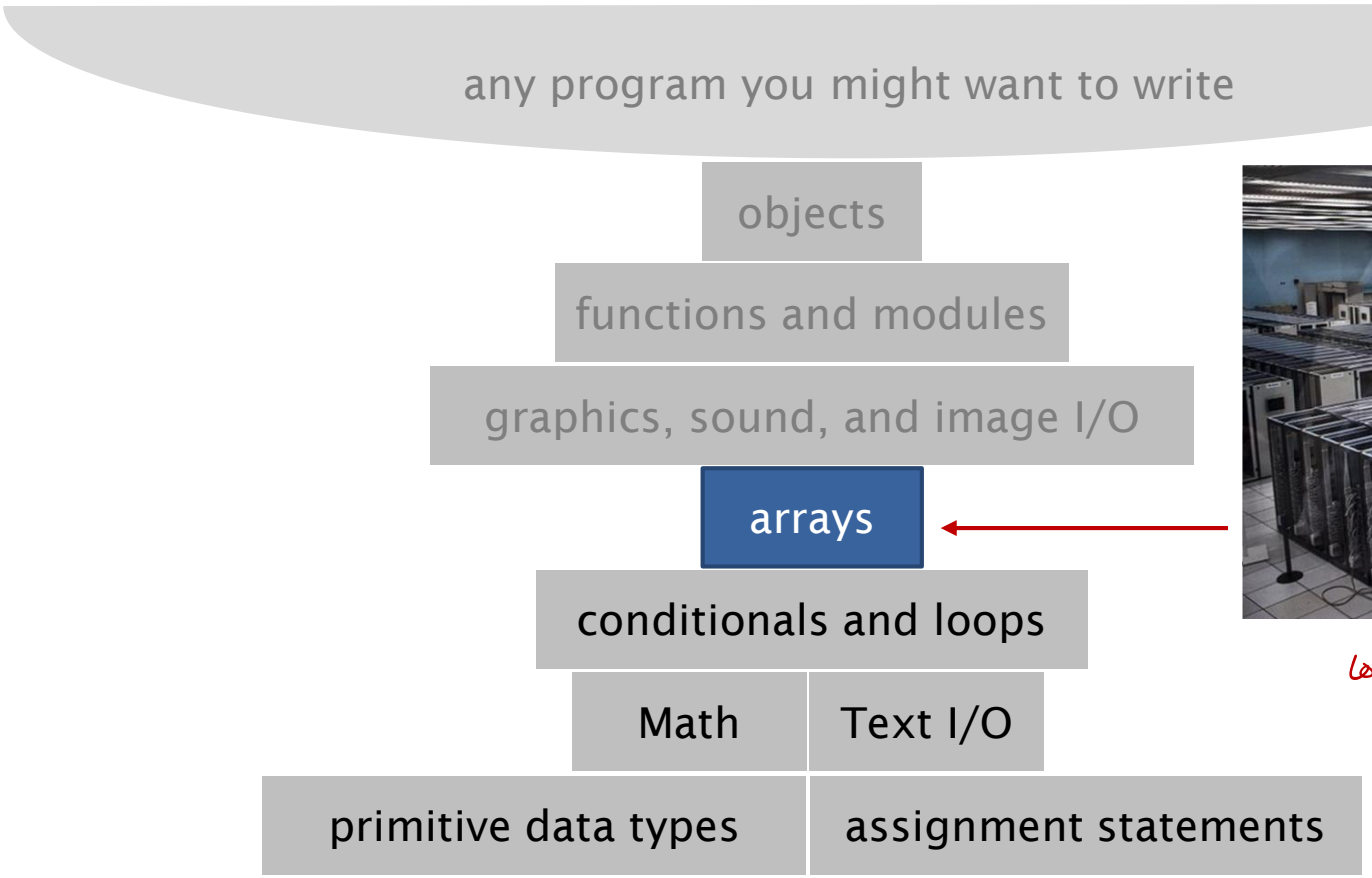
عناصر برنامه‌نویسی: آرایه‌ها

سید ناصر رضوی **www.snrazavi.ir**
۱۳۹۸

۱-۴ آرایه‌ها


INTRODUCTION TO
Programming
in Python
An Interdisciplinary Approach
Robert Sedgewick · Kevin Wayne · Robert Dondero

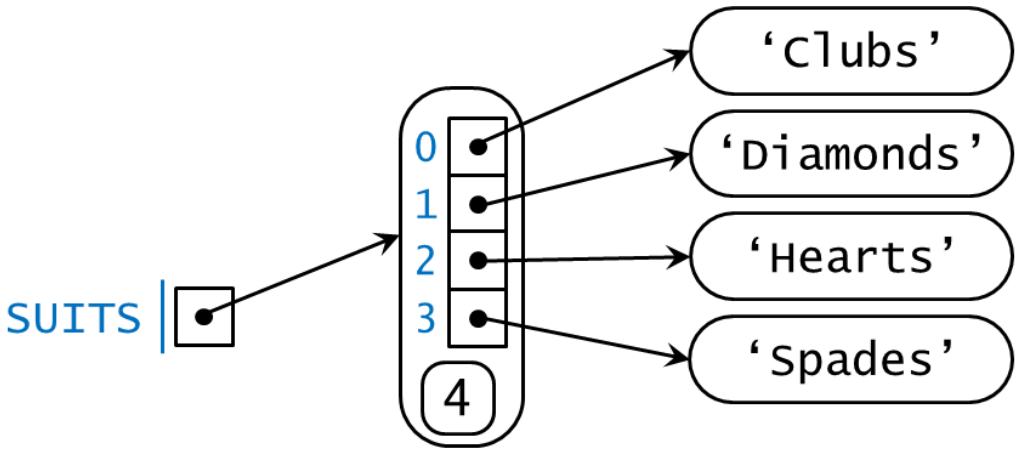
اجزای برنامه‌نویسی



توانایی ذخیره و پردازش حجم انبوهی از داده‌ها

آرایه‌ها

- ساختمان داده.
  - روش ذخیره کردن داده‌ها در حافظه (به منظور دسترسی و پردازش آسان‌تر و کارآتر داده‌ها)
- آرایه. یک ساختمان داده به منظور ذخیره‌سازی یک دنباله از (ارجاع‌ها به) اشیا
  - برای دسترسی به عناصر آرایه از شماره‌گذاری و اندیس‌گذاری استفاده می‌کنیم.



ساختمان داده آرایه

آرایه‌ها

- آرایه. یک دنباله اندیس‌گذاری شده از (ارجاع به) اشیا.

index	value
0	2♥
1	6♠
2	A♦
3	A♥
...	
49	3♣
50	K♣
51	4♠

- مثال‌ها.
  - ۵۲ کارت بازی در یک دسته ورق.
  - ۱۰ هزار دانشجوی کارشناسی در دانشگاه تبریز
  - ۱ میلیون پیکسل در یک تصویر دیجیتال
  - ۴ میلیارد نوکلئوتید در یک رشته DNA
  - ۷۳ میلیارد پرس و جوی گوگل در یک سال
  - ۸۶ میلیاد نورون در مغز
  - ۵۰ تریلیون سلول در بدن انسان

پردازش تعداد زیادی مقدار از یک نوع

۱۰ مقدار، بدون استفاده از آرایه

```
# tedious and error-prone
a0 = 0.0
a1 = 0.0
a2 = 0.0
a3 = 0.0
a4 = 0.0
a5 = 0.0
a6 = 0.0
a7 = 0.0
a8 = 0.0
a9 = 0.0
…
a4 = 3.0
…
a8 = 8.0
…
x = a4 + a8
```

۱۰ مقدار، با استفاده از آرایه

```
# easy alternative
a = [0] * 10
…
a[4] = 3.0
…
a[8] = 8.0
…
x = a[4] + a[8]
```

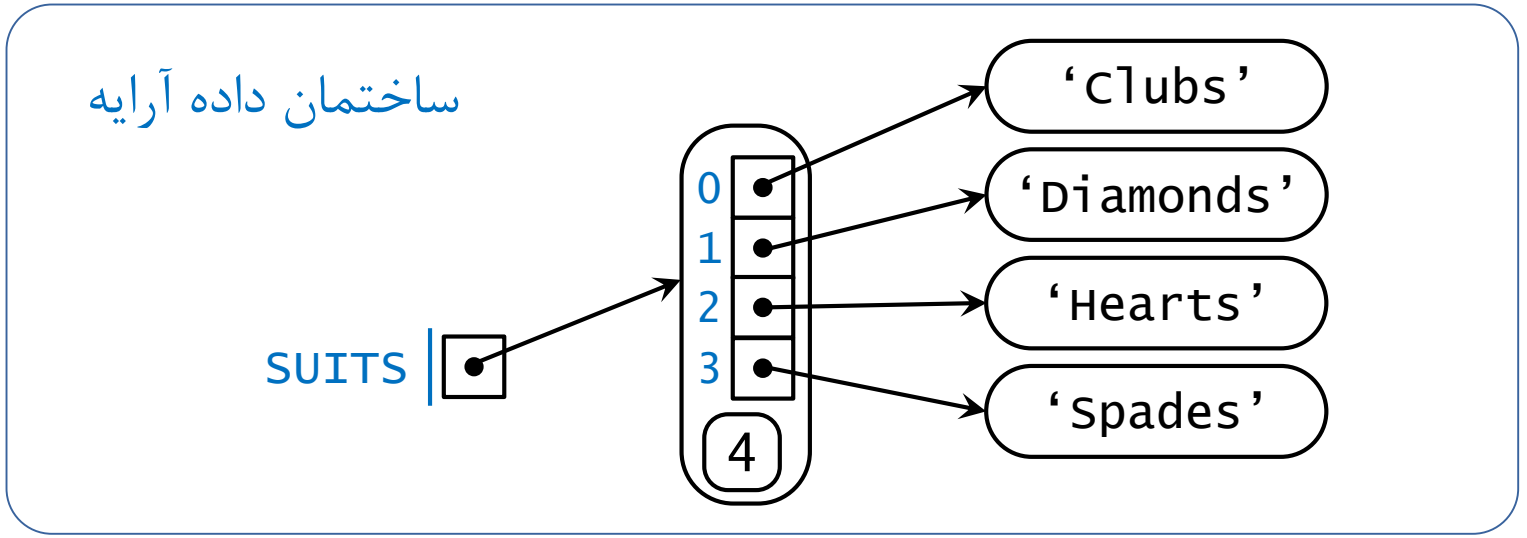
یک میلیون مقدار، با استفاده از آرایه

```
# handle huge amounts of data
a = [0] * 1000000
…
a[123456] = 3.0
…
a[987654] = 8.0
…
x = a[123456] + a[987654]
```

آرایه‌ها در پایتون

- ساده‌ترین روش ایجاد آرایه‌ها در پایتون.

```
SUITS = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
x = [0.30, 0.60, 0.10]
y = [0.50, 0.10, 0.40]
```



ضرب نقطه‌ای دو بردار

- ضرب نقطه‌ای. ضرب نقطه‌ای دو بردار x و y هر یک به طول n، برابر است با مجموع حاصل‌ضرب عناصر متناظر در دو بردار.

```
x = [0.30, 0.60, 0.10]
y = [0.50, 0.10, 0.40]

n = len(x)
total = 0
for i in range(n):
    total += x[i] * y[i]
```

i	x[i]	y[i]	x[i]*y[i]	total
				0.00
0	0.30	0.50	0.15	0.15
1	0.60	0.10	0.06	0.21
2	0.10	0.40	0.04	0.25
				0.25

کار کردن با آرایه‌ها در پایتون

- افزودن عناصر جدید به آرایه.

```
a = []
for i in range(n):
    a += [0.0]
```

ایجاد یک آرایه از اعداد اعشاری به طول n و مقداردهی عناصر آن با صفر

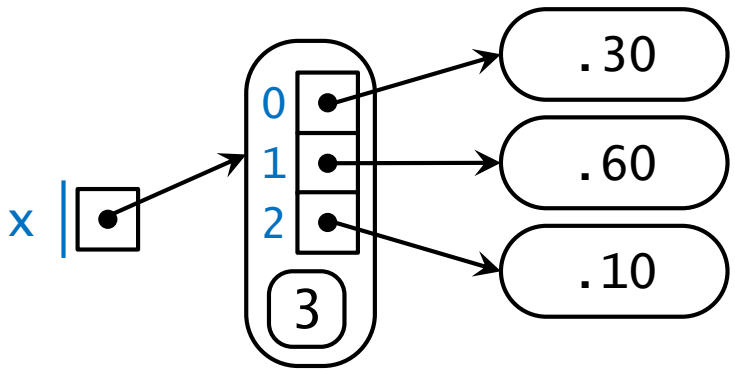
```
a = [0.0] * n
```

روش میان‌بر

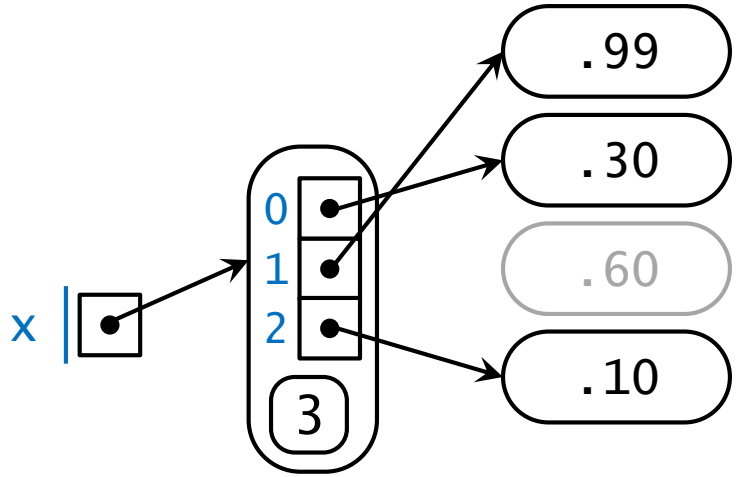
کار کردن با آرایه‌ها در پایتون

- آرایه یک شی تغییرپذیر است.
  - یعنی، می‌توانیم عناصر آرایه را تغییر دهیم.

```
x = [0.30, 0.60, 0.10]
```




```
x[1] = 0.99
```



کار کردن با آرایه‌ها در پایتون

- وارون کردن ترتیب عناصر یک آرایه.

```
n = len(a)
for i in range(n // 2):
    temp = a[i]
    a[i] = a[n - 1 - i]
    a[n - 1 - i] = temp
```

```
a = a[::-1]
```

	a[]						
i	0	1	2	3	4	5	6
	3	1	4	1	5	9	2
0	2	1	4	1	5	9	3
1	2	9	4	1	5	1	3
2	2	9	5	1	4	1	3
3	2	9	5	1	4	1	3
	2	9	5	1	4	1	3

کار کردن با آرایه‌ها در پایتون

- حلقه زدن بر روی عناصر یک آرایه.

محاسبه میانگین عناصر یک آرایه

```
total = 0.0

for v in a:
    total += v

average = total / len(a)
```

محاسبه میانگین عناصر یک آرایه

```
total = 0.0

for i in range(len(a)):
    total += a[i]

average = total / len(a)
```

کار کردن با آرایه‌ها در پایتون

- توابع پیش‌ساخته.
  - طول آرایه، محاسبه مجموع، کمینه، بیشینه.

تابع	توضیحات
`len`	تعداد عناصر آرایه
`sum`	مجموع عناصر آرایه
`min`	مقدار کمینه آرایه
`max`	مقدار بیشینه آرایه

محاسبه میانگین عناصر یک آرایه

```
total = 0.0

for i in range(len(a)):
    total += a[i]

average = total / len(a)
```

```
average = float(sum(a)) / len(a)
```

کار کردن با آرایه‌ها در پایتون

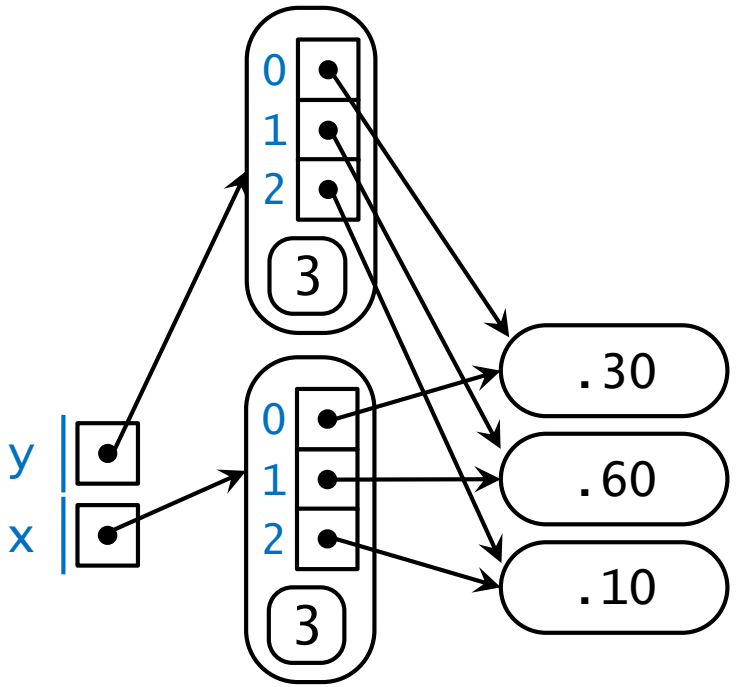
- کپی کردن یک آرایه.

ایجاد یک کپی از آرایه

```
x = [0.30, 0.60, 0.10]

y = []
for v in x:
    y += [v]
```

ایجاد یک کپی از آرایه با استفاده از عملگر برش

```
y = x[:]  # create a copy of x
```

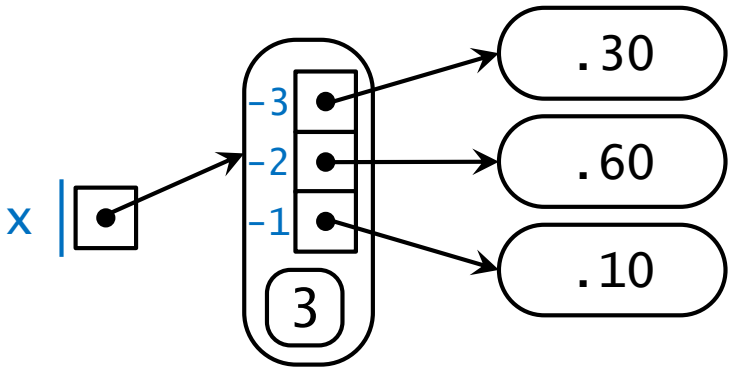


کار کردن با آرایه‌ها در پایتون

- عملگر برش.
  - ایجاد یک آرایه جدید شامل عناصر `a[i]` تا `a[j - 1]`.
    - مقدار پیش‌فرض `i` برابر است با `0`
    - مقدار پیش‌فرض `j` برابر است با `len(a)`

`a[i:j]`

- چند مثال.
  - `a[1:]`: یک آرایه جدید شامل همه عناصر به جز اولی.
  - `a[:-1]`: یک آرایه جدید شامل همه عناصر به جز آخری.
  - `a[:]`: یک آرایه جدید شامل همه عناصر



چند مثال از پردازش آرایه

create an array with random values	`a = []`
	`for i in range(n):`
	`    a += [random.random()]`
	`a = [random.random() for i in range(n)]`

print the array values, one per line	`for v in a:` `    print(v)`

find the maximum of the array values	`maximum = a[0]`
	`for v in a[1:]:`
	`    if (v > maximum):`
	`        maximum = v`
	`maximum = max(a)`

compute the average of the array values	`total = 0.0`
	`for v in a:`
	`    total += v`
	`average = total / len(a)`
	`average = float(sum(a)) / len(a)`

copy to another array	`b = []`
	`for v in a:`
	`    b += [v]`
	`b = a[:]`

reverse the elements within an array	`n = len(a)`
	`for i in range(n // 2):`
	`    temp = a[i]`
	`    a[i] = a[n - 1 - i]`
	`    a[n - 1 - i] = temp`
	`a = a[::-1]`

بر زدن یک دسته ورق

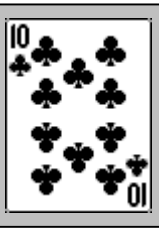
بر زدن یک دسته ورق

- بازنمایی کارت‌ها.

```
SUITS = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
RANKS = ['2', '3', '4', '5', '6', '7', '8', '9', '10',
         'Jack', 'Queen', 'King', 'Ace']
```

- انتخاب یک کارت به صورت تصادفی.

```
import random

rank = random.randrange(0, len(RANKS))
suit = random.randrange(0, len(SUITS))

print(RANKS[rank] + ' of ' + SUITS[suit])
```

'10 of clubs' ⟶ 

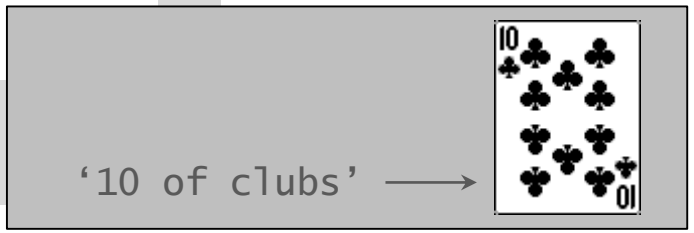
بر زدن یک دسته ورق

- بازنمایی کارت‌ها.

```
SUITS = ['Clubs', 'Diamonds', 'Hearts', 'Spades']
RANKS = ['2', '3', '4', '5', '6', '7', '8', '9', '10',
         'Jack', 'Queen', 'King', 'Ace']
```

- انتخاب یک کارت به صورت تصادفی.

```
import random

rank = random.choice(RANKS)
suit = random.choice(SUITS)

print(rank + ' of ' + suit)
```



بر زدن یک دسته ورق

- ایجاد یک دسته ورق و چاپ آنها در خروجی.

rank

0	1	2	3	4	5	6	7	8	9	10	11	12
2	3	4	5	6	7	8	9	10	J	Q	K	A

suit

0	1	2	3
♣	♦	♥	♠

```
deck = []
for rank in RANKS:
    for suit in SUITS:
        card = rank + ' of ' + suit
        deck += [card]
```

- پرسش. کارت‌ها به چه ترتیبی چاپ می‌شوند؟

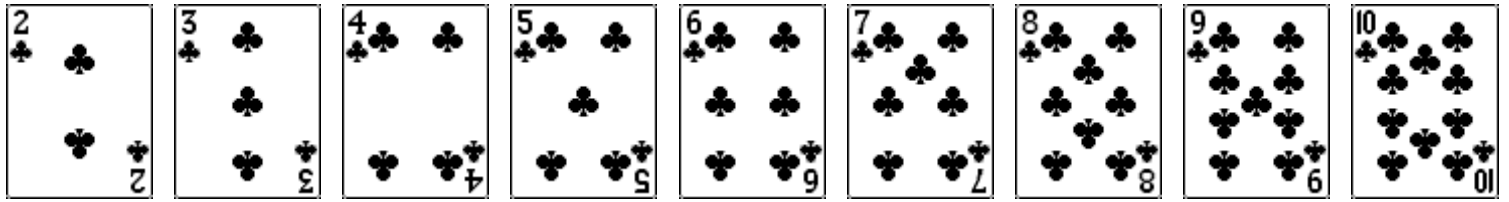
A.
```
2 of Clubs
2 of Diamonds
2 of Hearts
2 of Spades
3 of Clubs
...
```

B.
```
2 of Clubs
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
...
```

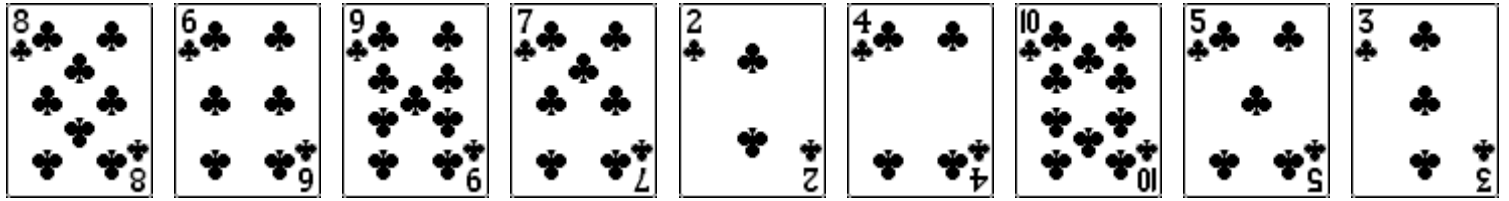
بر زدن یک دسته ورق

- بر زدن. بازآرایی عناصر یک آرایه به گونه‌ای که نتیجه یک جایگشت تصادفی با توزیع یکنواخت باشد.

پیش از بر زدن

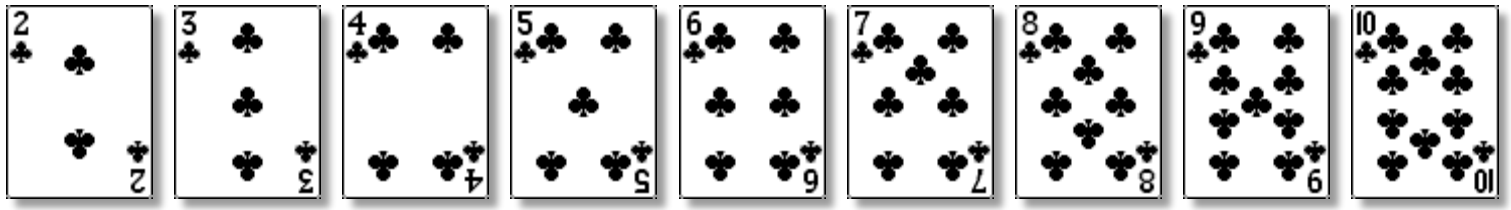


پس از بر زدن



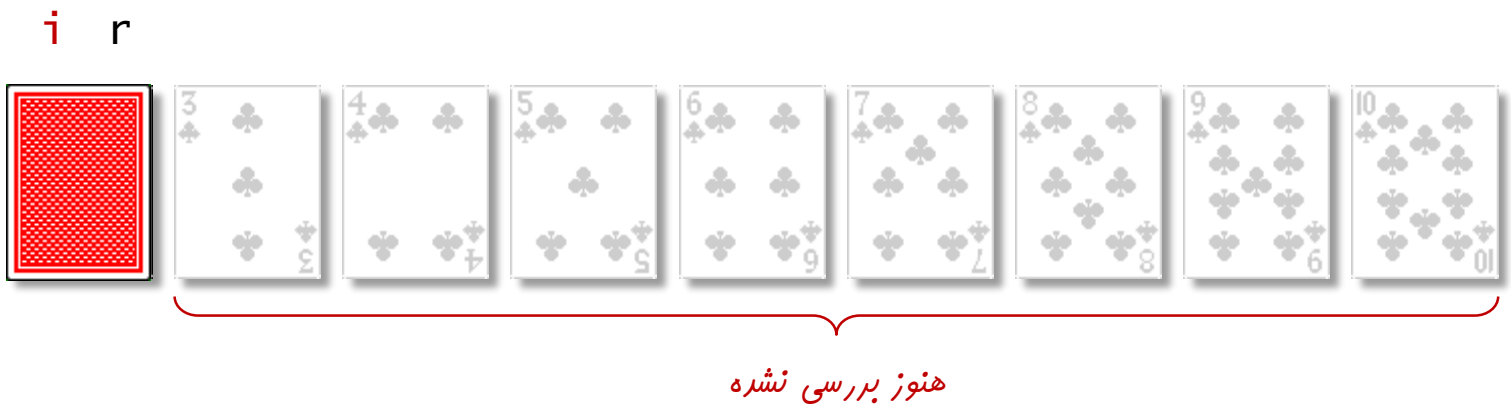
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



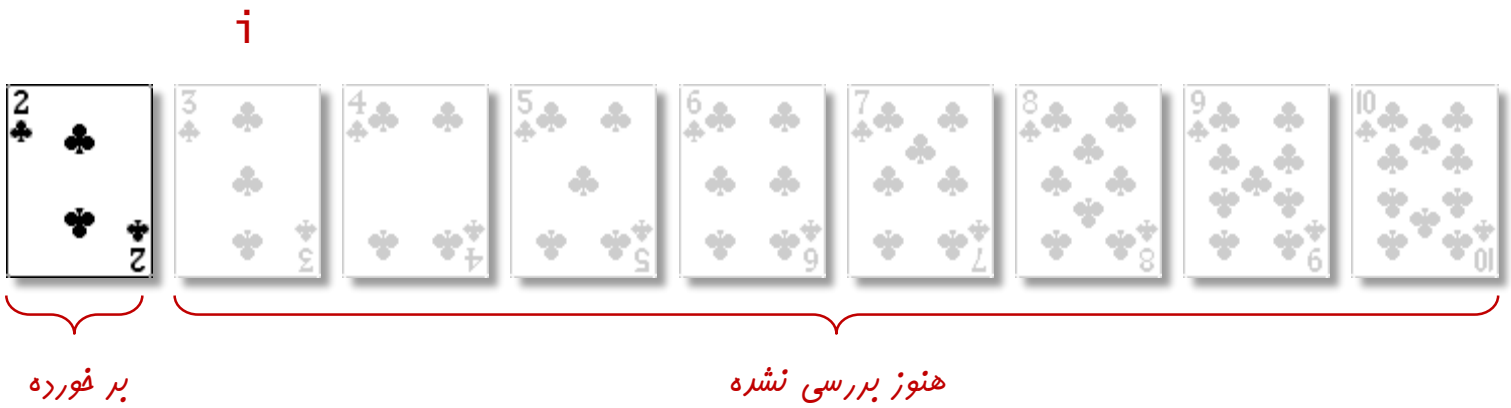
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



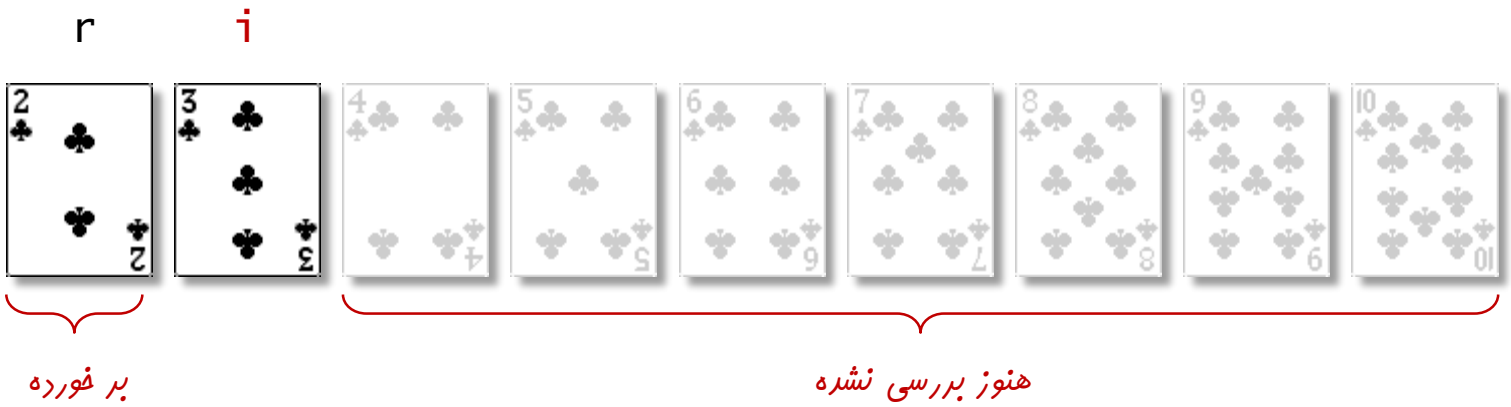
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



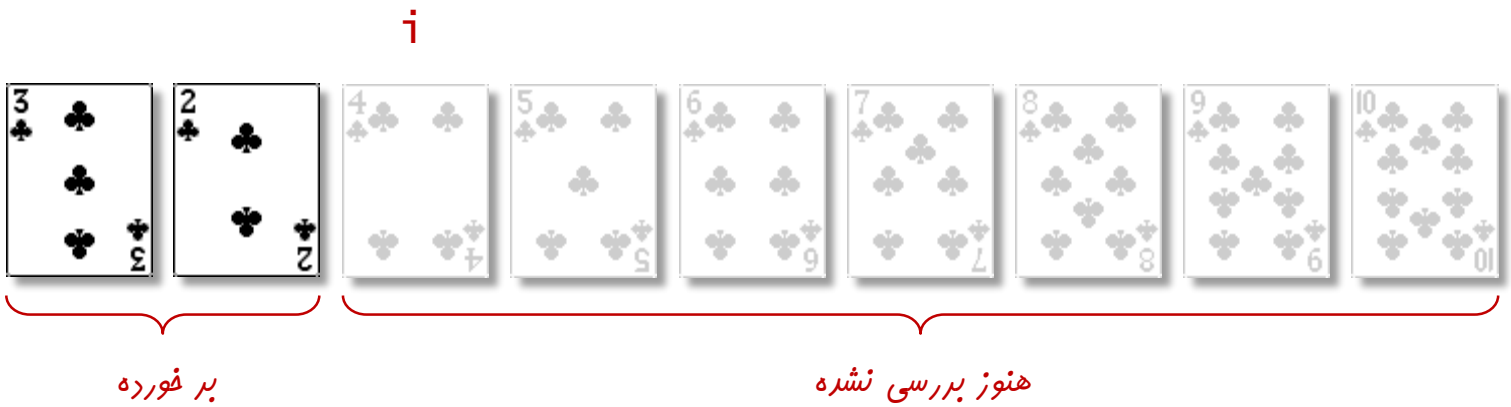
بر زدن یک دسته ورق

- در تکرار i یک عدد تصادفی مانند r بین صفر و i با احتمال مساوی تولید کن.
- کارت‌های a[i] و a[r] را جابه‌جا کن.



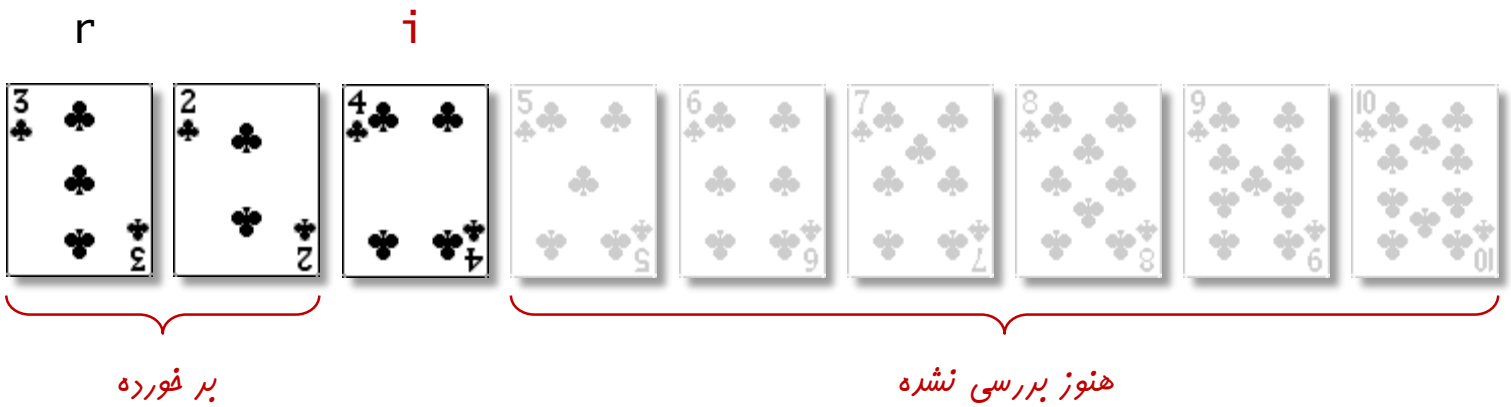
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



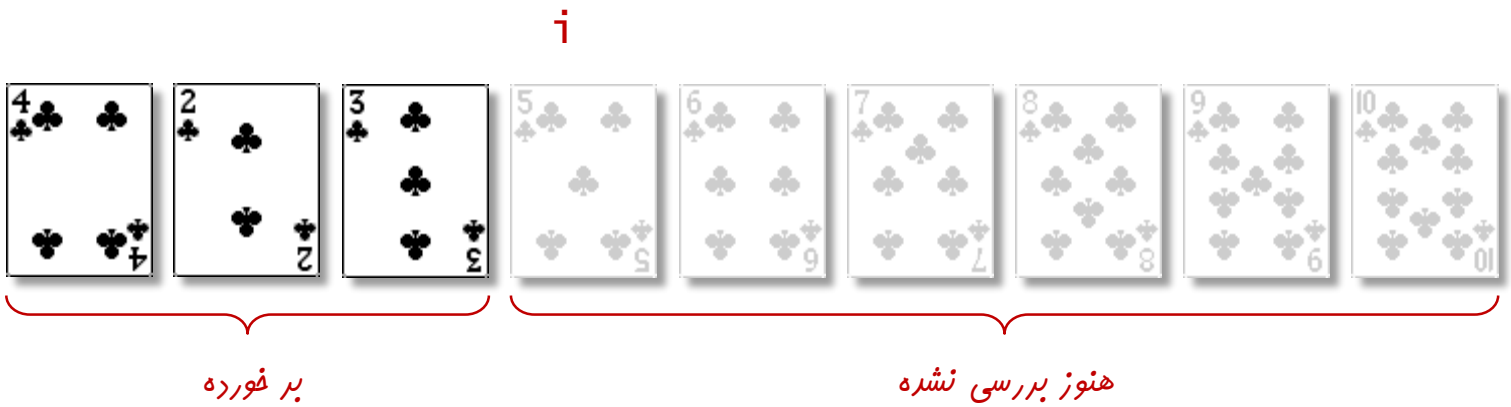
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



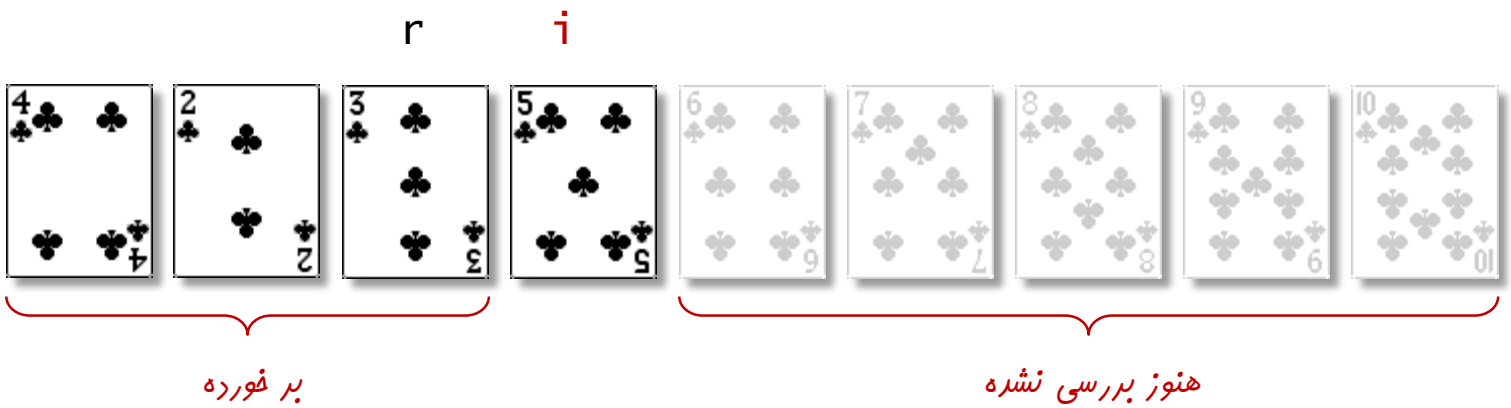
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



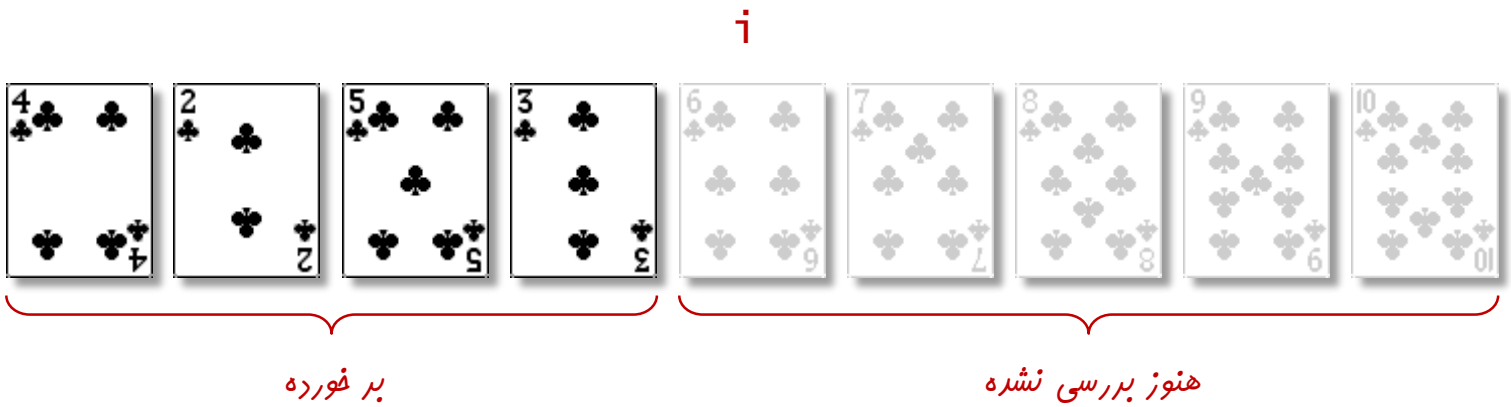
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



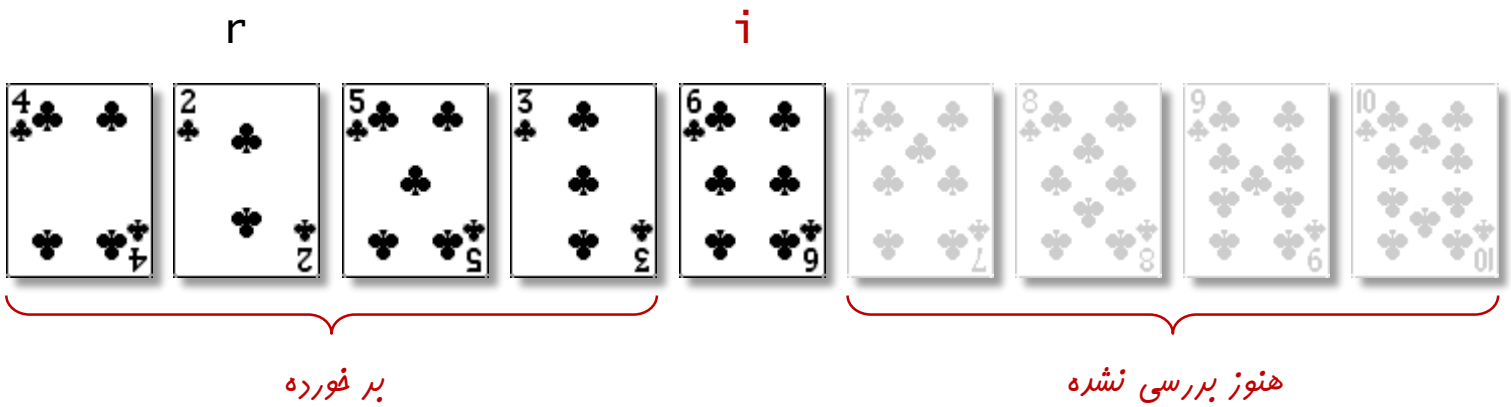
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



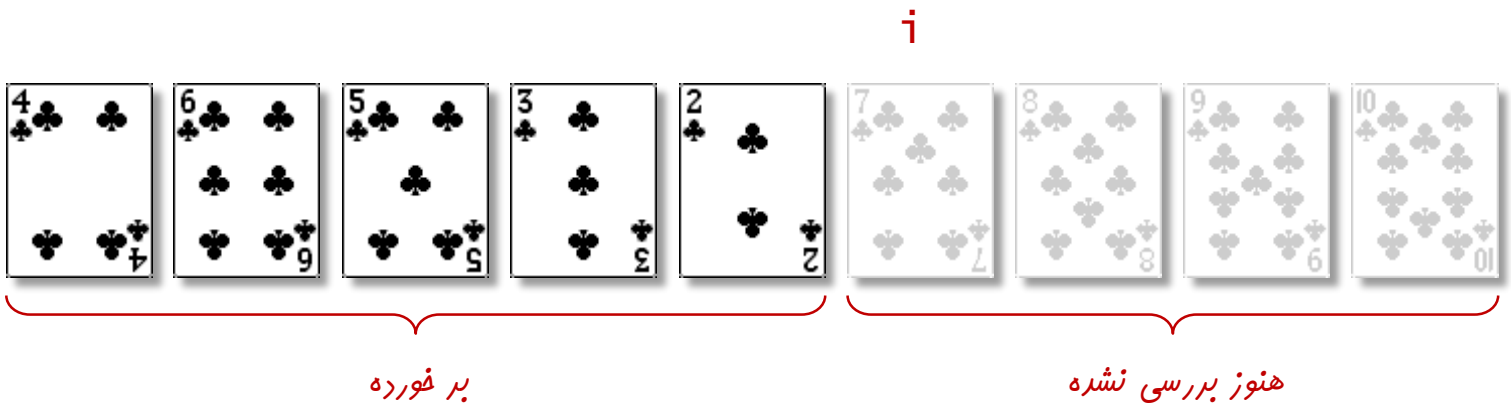
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



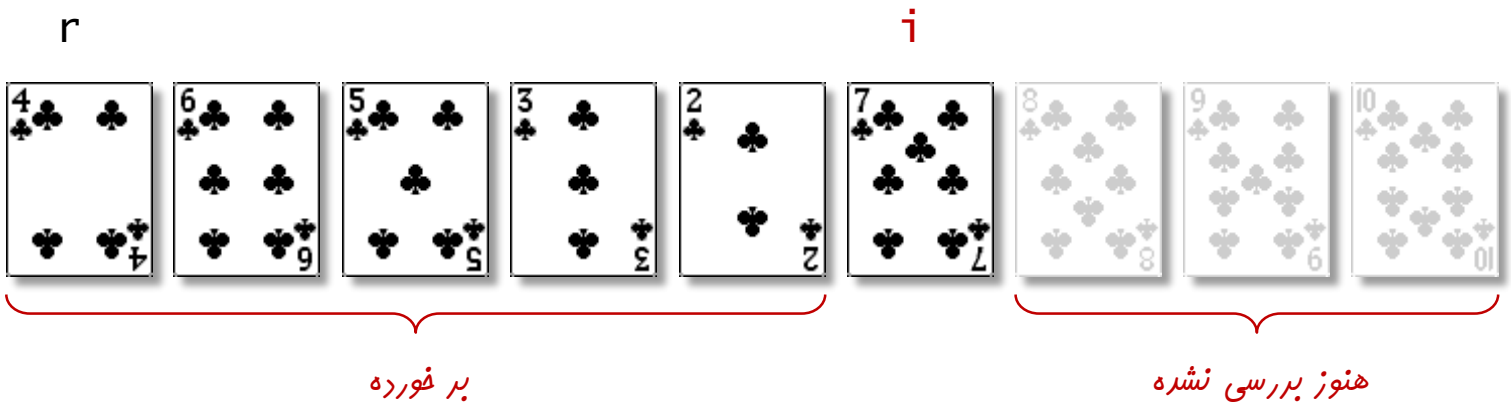
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



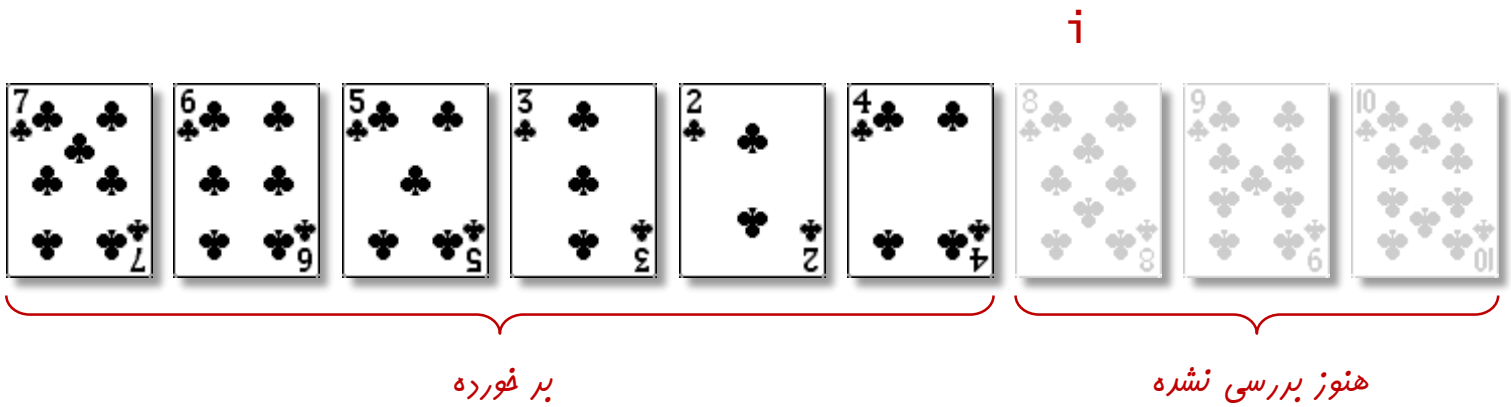
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



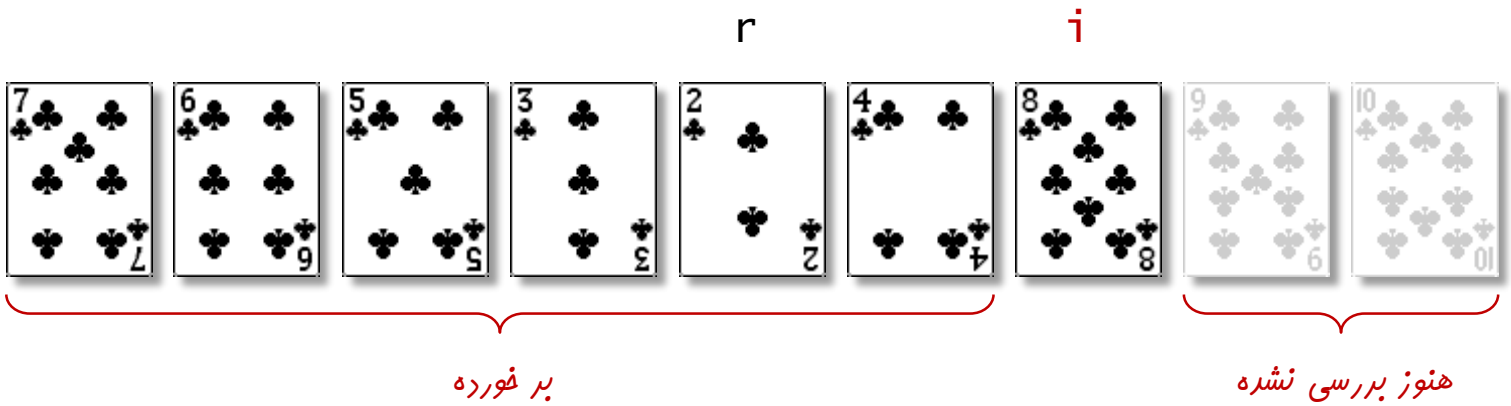
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



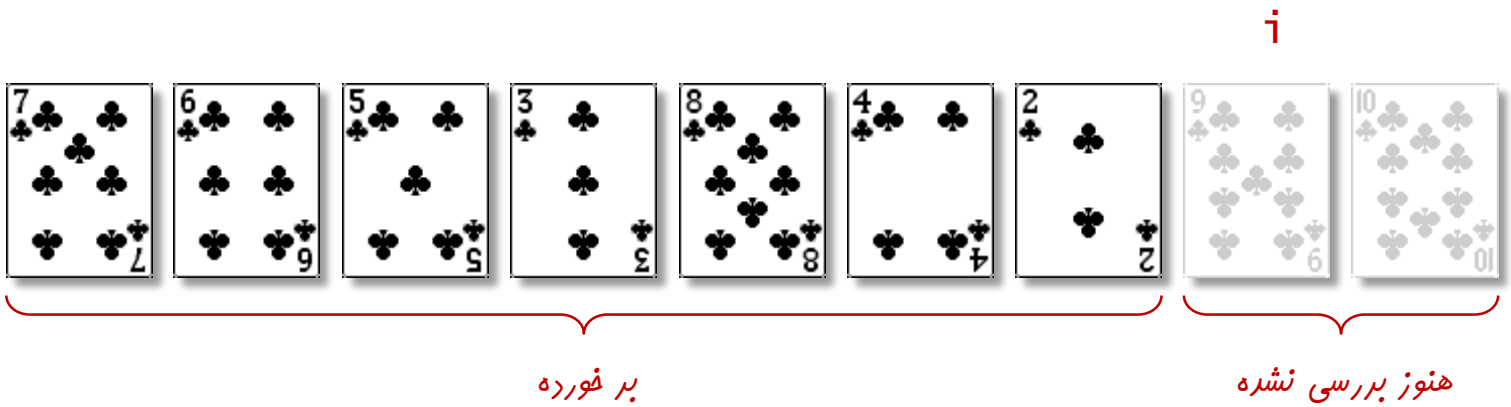
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



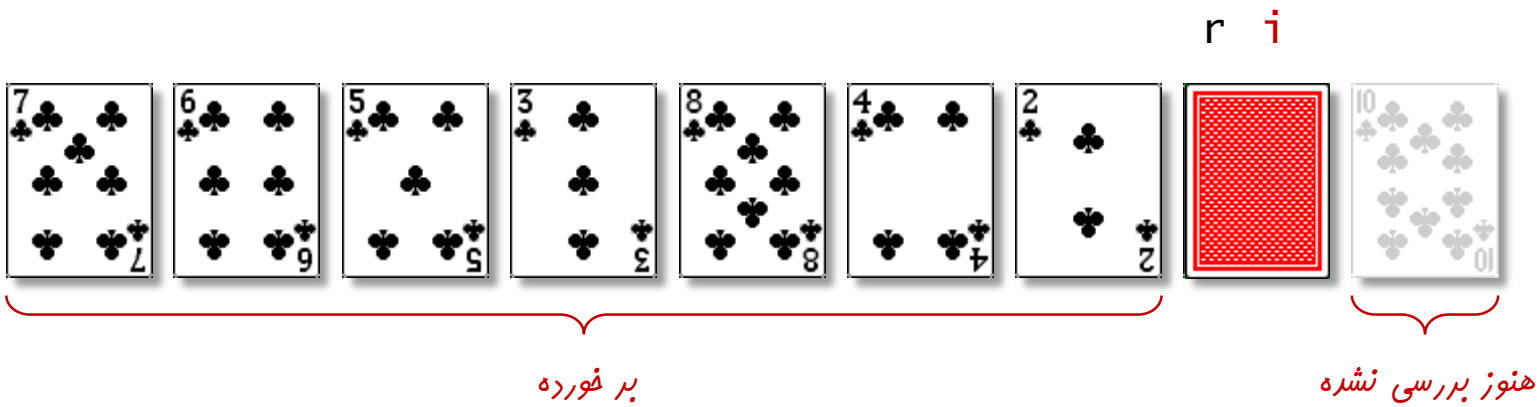
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



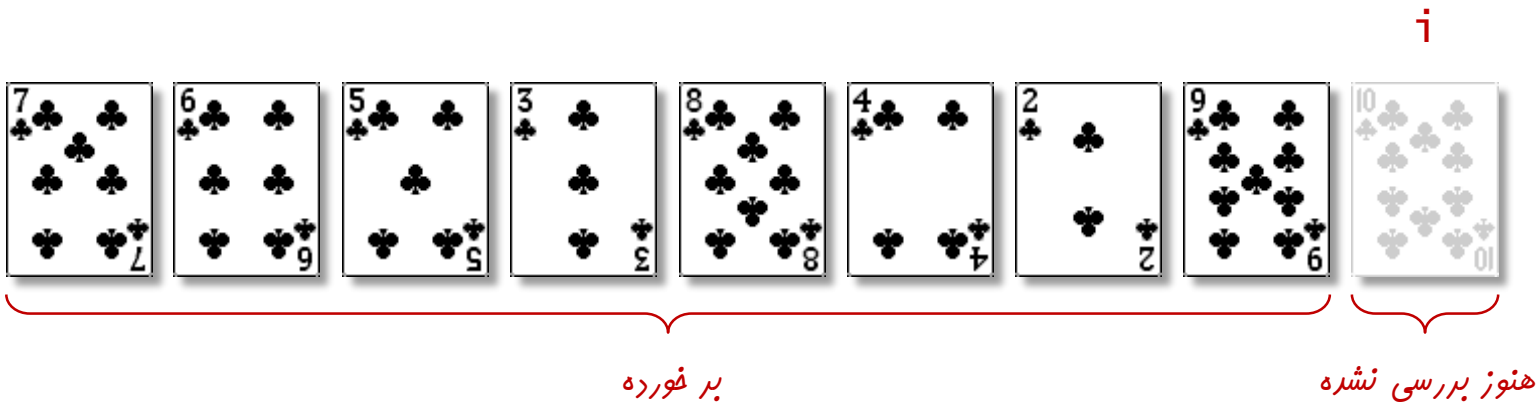
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



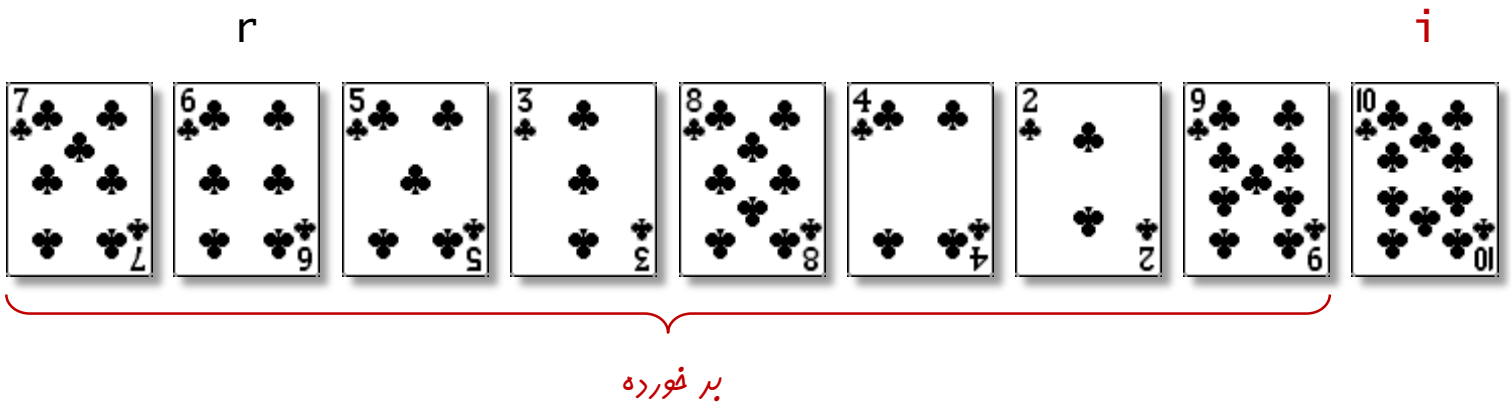
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



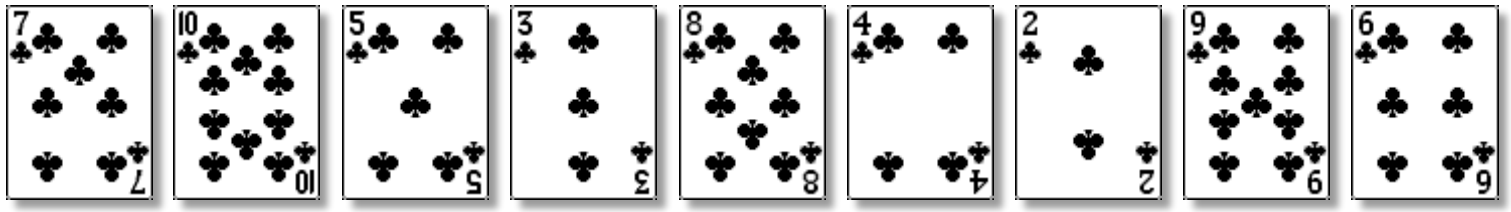
بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



بر زدن یک دسته ورق

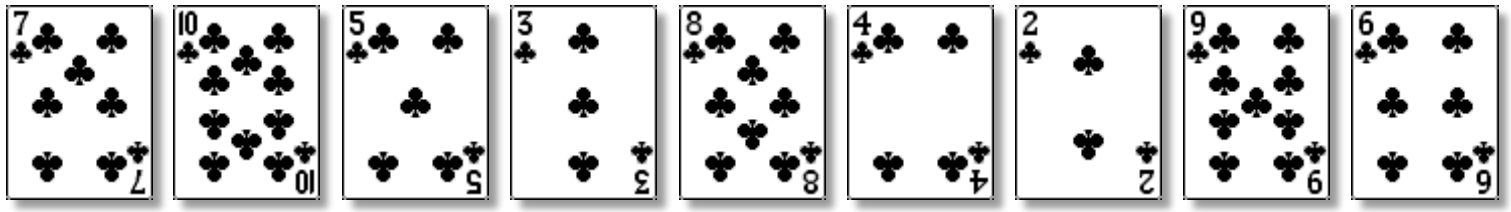
- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



بر خورده

بر زدن یک دسته ورق

- در تکرار `i` یک عدد تصادفی مانند `r` بین صفر و `i` با احتمال مساوی تولید کن.
- کارت‌های `a[i]` و `a[r]` را جابه‌جا کن.



- گزاره. [فیشر، ۱۹۳۸] الگوریتم بر زدن کنوث در زمان خطی یک جایگشت تصادفی یکنواخت از آرایه ورودی تولید می‌کند.

بر زدن یک دسته ورق

- هدف. با داشتن یک آرایه، عناصر آن را به یک ترتیب تصادفی بازآرایی کن.

- الگوریتم بر زدن.
  - در تکرار i، یک کارت تصادفی از بین کارت‌های deck[0] تا deck[i] انتخاب کن.
  - کارت تصادفی انتخاب شده را با deck[i] جابه‌جا کن.

```
n = len(deck)
for i in range(n):
    r = random.randrange(0, i + 1)
    temp = deck[r]
    deck[r] = deck[i]
    deck[i] = temp
```

بر زدن یک دسته ورق


```
% python deck.py
10 of Diamonds
King of Spades
2 of Spades
3 of Clubs
4 of Spades
Queen of Clubs
2 of Hearts
7 of Diamonds
6 of Spades
Queen of Spades
3 of Spades
Jack of Diamonds
6 of Diamonds
8 of Spades
9 of Diamonds
...
10 of Spades
```

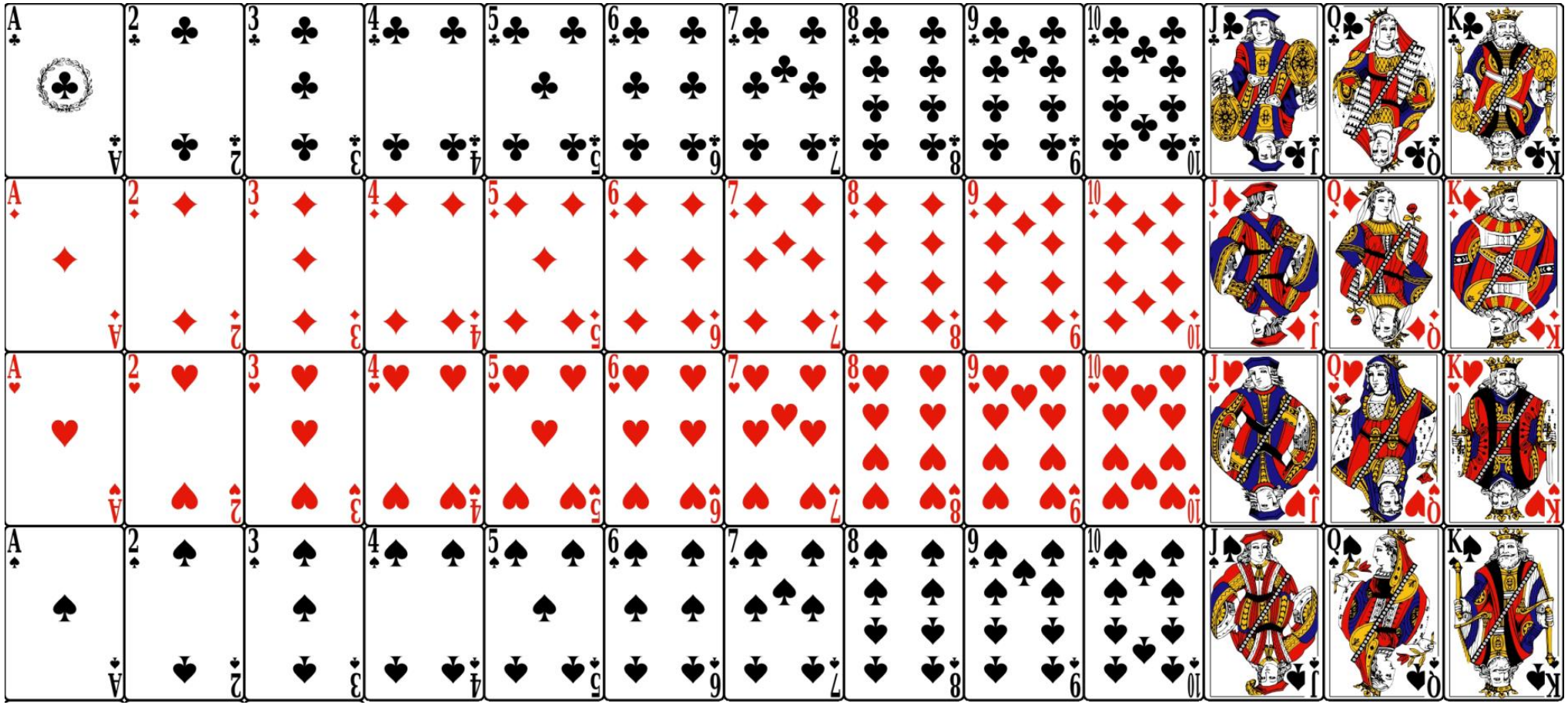
```
% python deck.py
5 of Clubs
Jack of Hearts
9 of Spades
10 of Spades
9 of Clubs
7 of Spades
6 of Diamonds
7 of Hearts
7 of Clubs
4 of Spades
Queen of Diamonds
10 of Hearts
5 of Diamonds
Jack of Clubs
Ace of Hearts
...
5 of Spades
```

انتخاب بدون تکرار

```python
import sys
import random

m = int(sys.argv[1])
n = int(sys.argv[2])

perm = [i for i in range(n)]  # comprehension

for i in range(m):
    r = random.randrange(i, n)
    perm[r], perm[i] = perm[i], perm[r] # swap

for i in range(m):
    print(str(perm[i]), end=' ')
print()
```

```
% python sample.py 6 16
9 6 3 5 4 13

% python sample.py 10 1000
566 240 662 680 500 491 309 348 563 34

% python sample.py 10 10
6 5 1 4 7 3 2 9 0 8
```

مسئله جمع‌کننده کالابرگ‌ها



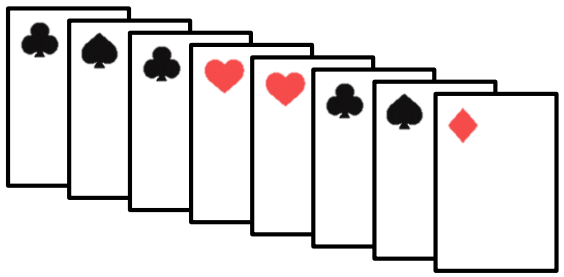
مسئله جمع‌کننده کالابرگ‌ها

- مسئله جمع‌کننده کالابرگ‌ها. با داشتن n نوع کارت مختلف، برای این که از هر نوع کارت (حداقل) یکی داشته باشیم چند کارت باید انتخاب شود؟

فرض می‌کنیم در هر انتخاب، احتمال انتخاب کارت‌ها با یکدیگر برابر است.

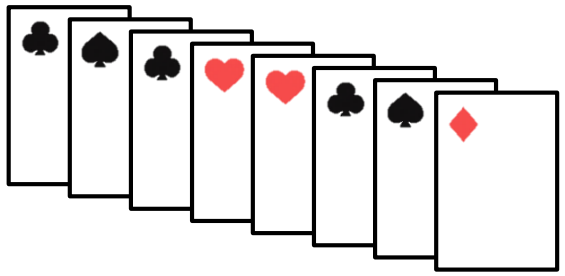
- مثال. جمع کردن همه ۴ نوع خال در یک دنباله تصادفی از کارت‌ها.

♣	♠	♥	♦
9♣	5♠	8♥	10♦
8♣	A♠	10♥	
K♣			



مسئله جمع‌کننده کالابرگ‌ها

- مسئله جمع‌کننده کالابرگ‌ها. با داشتن n نوع کارت مختلف، برای این که از هر نوع کارت (حداقل) یکی داشته باشیم چند کارت باید انتخاب شود؟



فرض می‌کنیم در هر انتخاب، احتمال انتخاب کارت‌ها با یکدیگر برابر است.

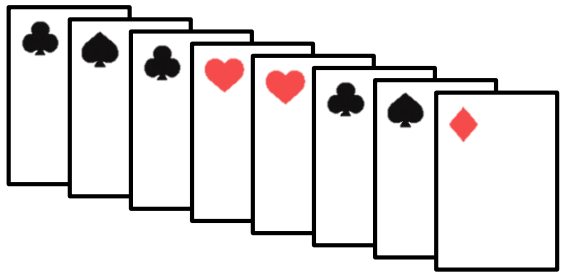
- مثال. جمع کردن همه ۱۳ نوع کارت در یک دنباله تصادفی از کارت‌ها.

9♣	5♠	8♥	10♦	2♠	A♠	10♥	Q♦	3♠	9♥	5♦	9♣	7♦	2♦	8♣	6♣	Q♥	K♣	10♥	A♦	4♦	J♥

2	3	4	5	6	7	8	9	10	J	Q	K	A
2♠	3♠	4♦	5♠	6♣	7♦	8♥	9♣	10♦	J♥	Q♦	K♣	A♠
2♦			5♦			8♣	9♥	10♥		Q♥		A♦
							9♣	10♥				

مسئله جمع‌کننده کالابرگ‌ها

- مسئله جمع‌کننده کالابرگ‌ها. با داشتن n نوع کارت مختلف، برای این که از هر نوع کارت (حداقل) یکی داشته باشیم چند کارت باید انتخاب شود؟



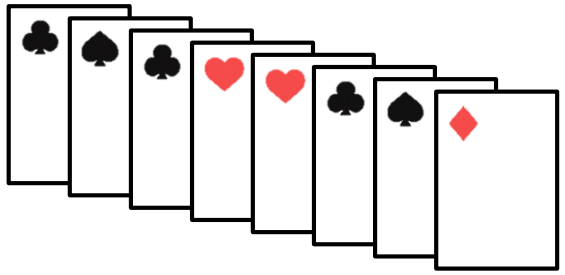
فرض می‌کنیم در هر انتخاب، احتمال انتخاب کارت‌ها با یکدیگر برابر است.

- الگوریتم شبیه‌سازی. به صورت تکراری هر بار یک عدد صحیح بین 0 و n - 1 انتخاب کن، تا زمانی که از هر کارت حداقل یکی داشته باشیم.

- پرسش. چگونه می‌توانیم بررسی کنیم که از هر نوع کارت حداقل یکی داریم؟
- پاسخ. با استفاده از یک آرایه بولی به گونه‌ای که `is_collected[i]` تنها وقتی درست است که از کارت نوع `i` حداقل یکی داشته باشیم.

مسئله جمع‌کننده کالابرگ‌ها

- مسئله جمع‌کننده کالابرگ‌ها. با داشتن n نوع کارت مختلف، برای این که از هر نوع کارت (حداقل) یکی داشته باشیم چند کارت باید انتخاب شود؟



فرض می‌کنیم در هر انتخاب، احتمال انتخاب کارت‌ها با یکدیگر برابر است.

- الگوریتم شبیه‌سازی. به صورت تکراری هر بار یک عدد صحیح بین 0 و n - 1 انتخاب کن، تا زمانی که از هر کارت حداقل یکی داشته باشیم.
- پرسش. چگونه می‌توانیم بررسی کنیم که از هر نوع کارت حداقل یکی داریم؟
- پاسخ. با استفاده از یک آرایه بولی به گونه‌ای که `is_collected[i]` تنها وقتی درست است که از کارت نوع `i` حداقل یکی داشته باشیم.

is_collected				
0	1	2	...	n - 1
F	F	F	...	F

مسئله جمع‌کننده کالابرگ‌ها


```
import sys
import random

n = int(sys.argv[2])

count = 0
collected_count = 0
is_collected = [False] * n

while collected_count < n:
    count += 1
    value = random.randrange(0, n)
    if not is_collected[value]:
        collected_count += 1
        is_collected[value] = True

print(count)
```

مسئله جمع‌کننده کالابرگ‌ها: اشکال‌زدایی

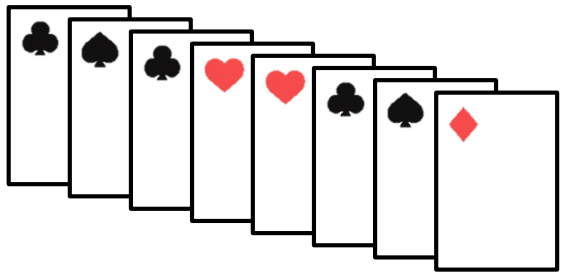
- اشکال‌زدایی.
  - افزودن کد به برنامه به منظور چاپ محتویات همه متغیرها.

value	is_collected 0	1	2	3	4	5	count	collected_count
	F	F	F	F	F	F	0	0
2	F	F	T	F	F	F	1	1
0	T	F	T	F	F	F	2	2
4	T	F	T	F	T	F	3	3
0	T	F	T	F	T	F	4	3
1	T	T	T	F	T	F	5	4
2	T	T	T	F	T	F	6	4
5	T	T	T	F	T	T	7	5
0	T	T	T	F	T	T	8	5
1	T	T	T	F	T	T	9	5
3	T	T	T	T	T	T	10	6

```
while collected_count < n:
    count += 1
    value = random.randrange(0, n)
    if not is_collected[value]:
        collected_count += 1
        is_collected[value] = True
```

مسئله جمع‌کننده کالابرگ‌ها: برخورد ریاضی

- مسئله جمع‌کننده کالابرگ‌ها. با داشتن n نوع کارت مختلف، برای این که از هر نوع کارت (حداقل) یکی داشته باشیم چند کارت باید انتخاب شود؟

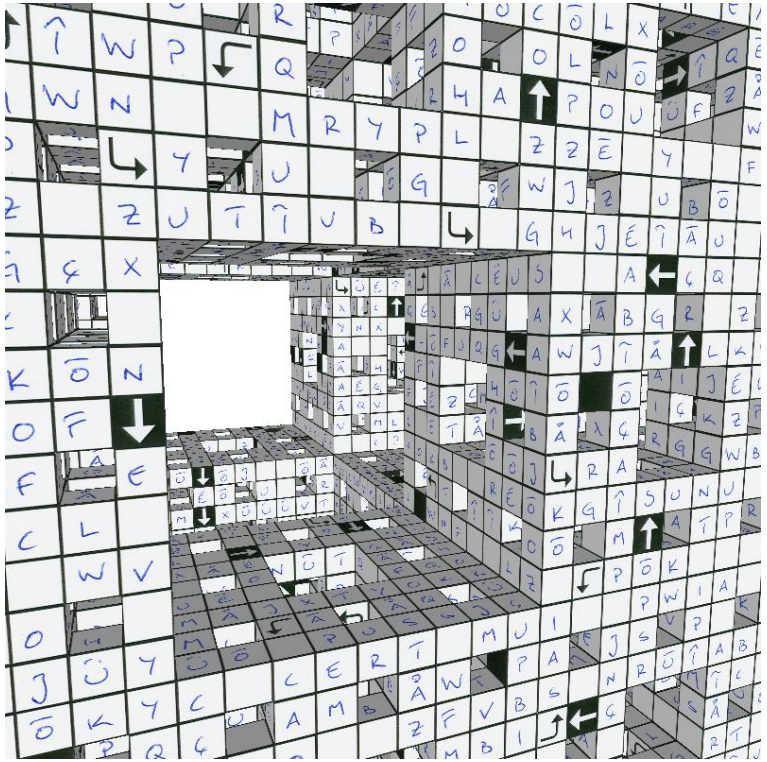


- حقیقت. تقریباً $n \ln n \sim n (1 + 1/2 + 1/3 + \ldots + 1/n)$

تحت شرایط ایده‌آل

- مثال. برای ۳۰ تیم بیسبال، تقریباً ۱۲۰ سال زمان نیاز داریم تا هر تیم حداقل یک بار قهرمان شود.

آرایه‌های چند بعدی

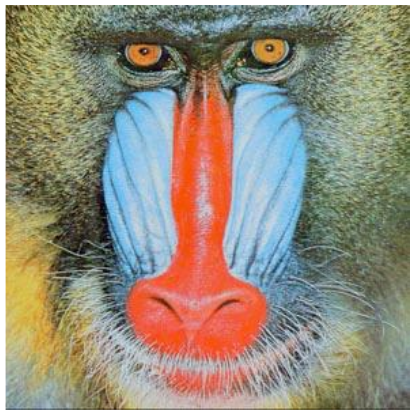


آرایه‌های دو بعدی

- آرایه‌های دو بعدی.
  - ماتریس‌ها در محاسبات ریاضی.
  - یک جدول از نمرات به ازای هر دانشجو و تمرینات.
  - یک جدول از داده‌ها به ازای هر آزمایش و نتایج آن.
  - تراکنش‌های مربوط به مشتریان یک بانک.
  - پیکسل‌ها در یک تصویر دیجیتال.
  - داده‌های جغرافیایی.

grade

student ID	0	1	2	3	4	5	…
0	A	A	C	B	A	C	
1	B	B	B	B	A	A	
2	C	D	D	B	C	A	
3	A	A	A	A	A	A	
4	C	C	B	C	B	B	
5	A	A	A	B	A	A	
…							



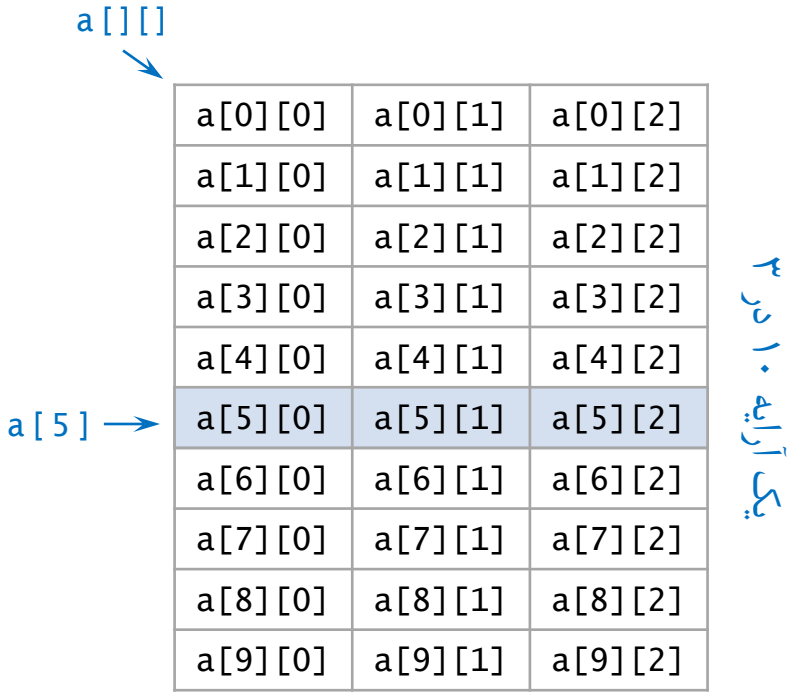
y-coordinate

x-coordinate

آرایه‌های دو بعدی در پایتون: ایجاد و مقداردهی

- دستیابی به عناصر. `a[i][j]` = عنصر واقع در سطر `i` و ستون `j`.
- اندیس گذاری مبتنی بر صفر. اندیس سطرها و ستون‌ها از صفر شروع می‌شود.

```
m = 10
n = 3

a = []
for i in range(m):
    row = [0] * n
    a += [row]
```

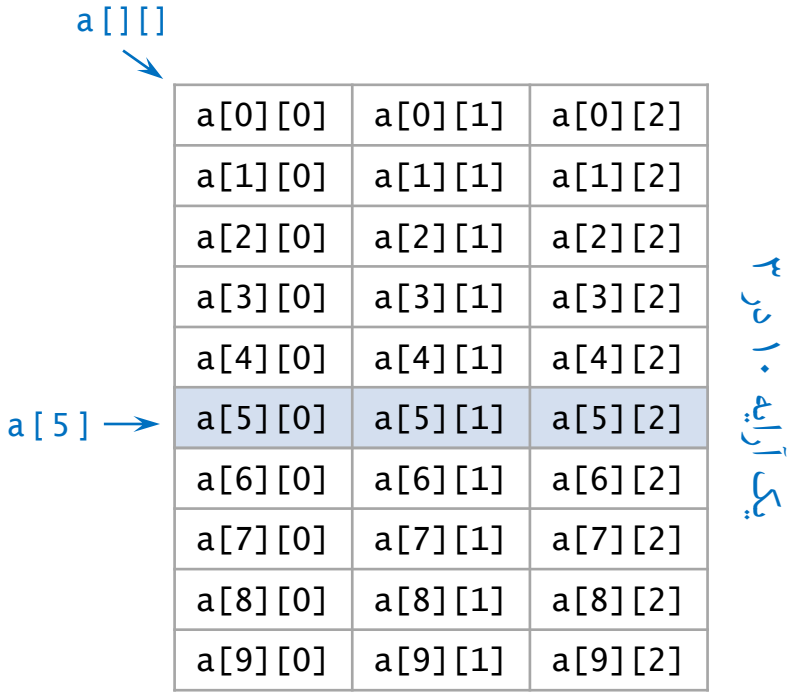


آرایه‌های دو بعدی در پایتون: دسترسی به عناصر

- دستیابی به عناصر. `a[i][j]` = عنصر واقع در سطر `i` و ستون `j`.
- اندیس گذاری مبتنی بر صفر. اندیس سطرها و ستون‌ها از صفر شروع می‌شود.

```
for i in range(m):
    for j in range(n):
        print(a[i][j], end=' ')
    print()
```

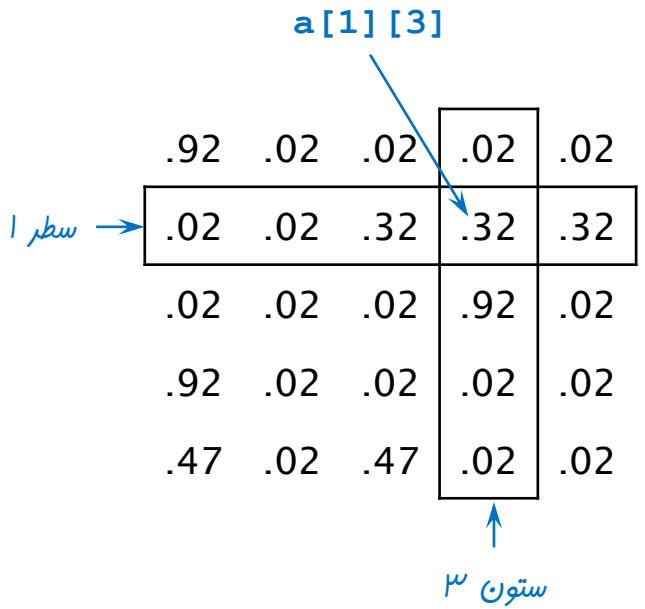
```
for row in a:
    for v in row:
        print(v, end=' ')
    print()
```



مقداردهی آرایه دو بعدی

- مقداردهی آرایه دو بعدی با لیست کردن مقادیر.

```
a = [[ .92, .02, .02, .02, .02 ],
     [ .02, .02, .32, .32, .32 ],
     [ .02, .02, .02, .92, .02 ],
     [ .92, .02, .02, .02, .02 ],
     [ .47, .02, .47, .02, .02 ]]
```

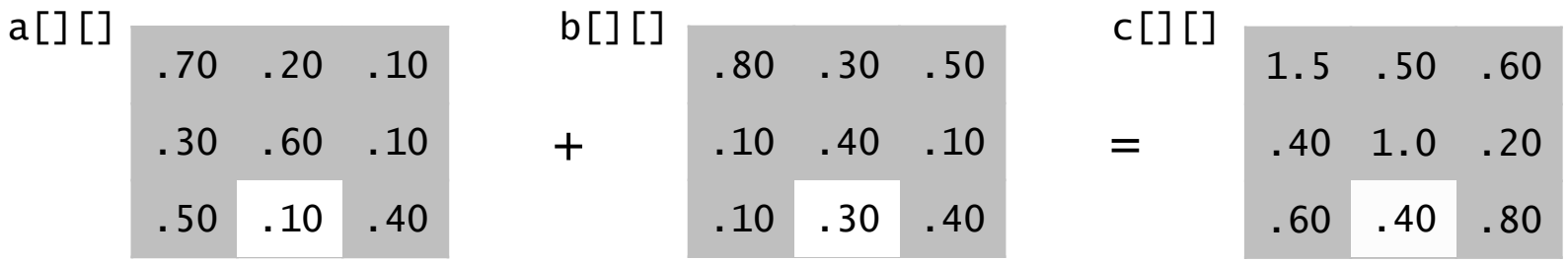


جمع ماتریسی

- جمع ماتریسی. با داشتن دو ماتریس a و b با ابعاد n در n، ماتریس c یک ماتریس n در n است به گونه‌ای که:

```
c[i][j] = a[i][j] + b[i][j]
```

```
for i in range(n):
    for j in range(n):
        c[i][j] = a[i][j] + b[i][j]
```

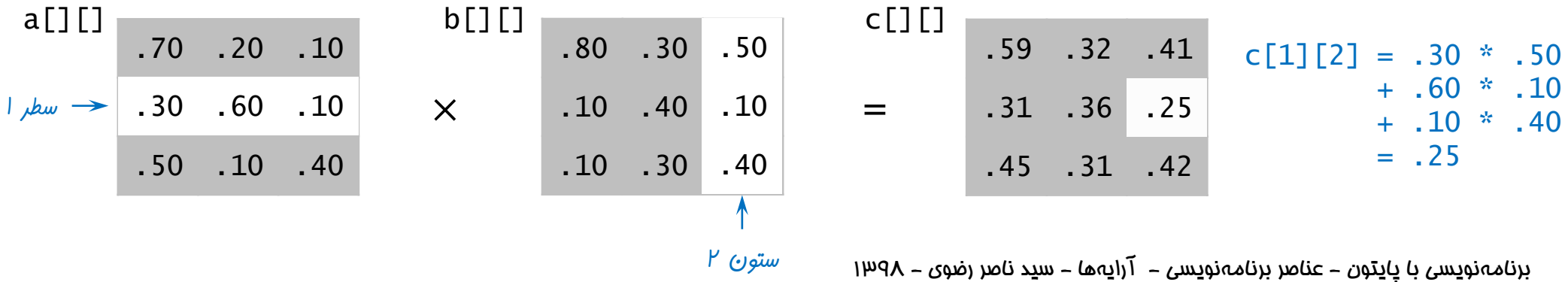


ضرب ماتریسی

- ضرب ماتریسی. با داشتن دو ماتریس a و b با ابعاد n در n، ماتریس c یک ماتریس n در n است به گونه‌ای که در آن c[i][j] برابر است با ضرب داخلی سطر i از ماتریس a[][] در ستون j از ماتریس b[][].

```
for i in range(n):
    for j in range(n):
        for k in range(n):
            c[i][j] += a[i][k] * b[k][j]
```

ضرب داخلی
سطر i از ماتریس a[][] در
ستون j از ماتریس b[][]



تحلیل ضرب ماتریسی

- پرسش. برای ضرب دو ماتریس n در n به چند عمل ضرب اسکالر نیاز است؟

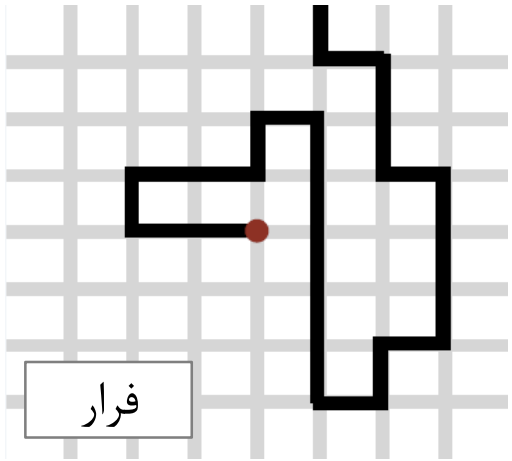
A. n B. n^2 C. n^3 D. n^4

```
for i in range(n):
    for j in range(n):
        for k in range(n):
            c[i][j] += a[i][k] * b[k][j]
```

مسئله ولگشت خودگریز

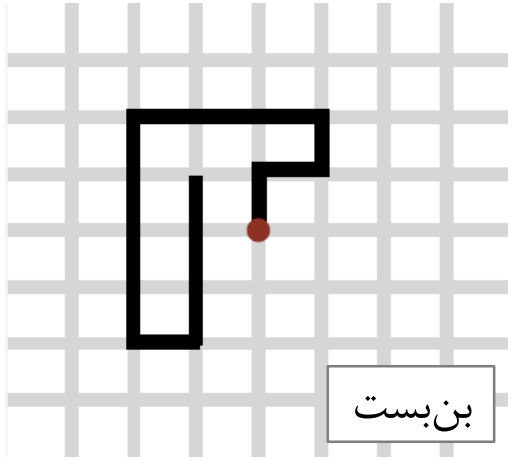
یک سگ به صورت تصادفی در حال گردش در یک شهر است و هرگز از یک تقاطع دو بار گذر نمی‌کند.

پرسش. آیا این سگ می‌تواند از شهر خارج شود؟



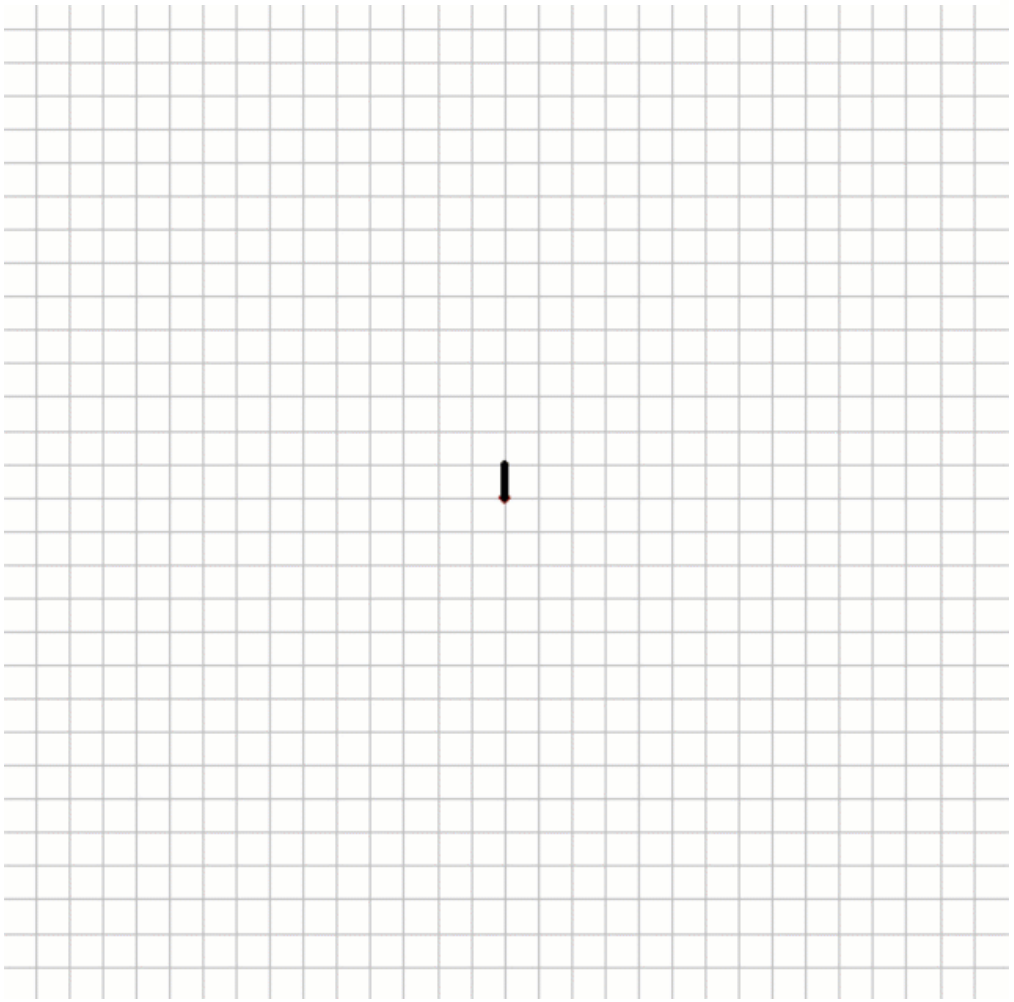
مدل. یک فرایند تصادفی در یک صفحه مشبک n در n.

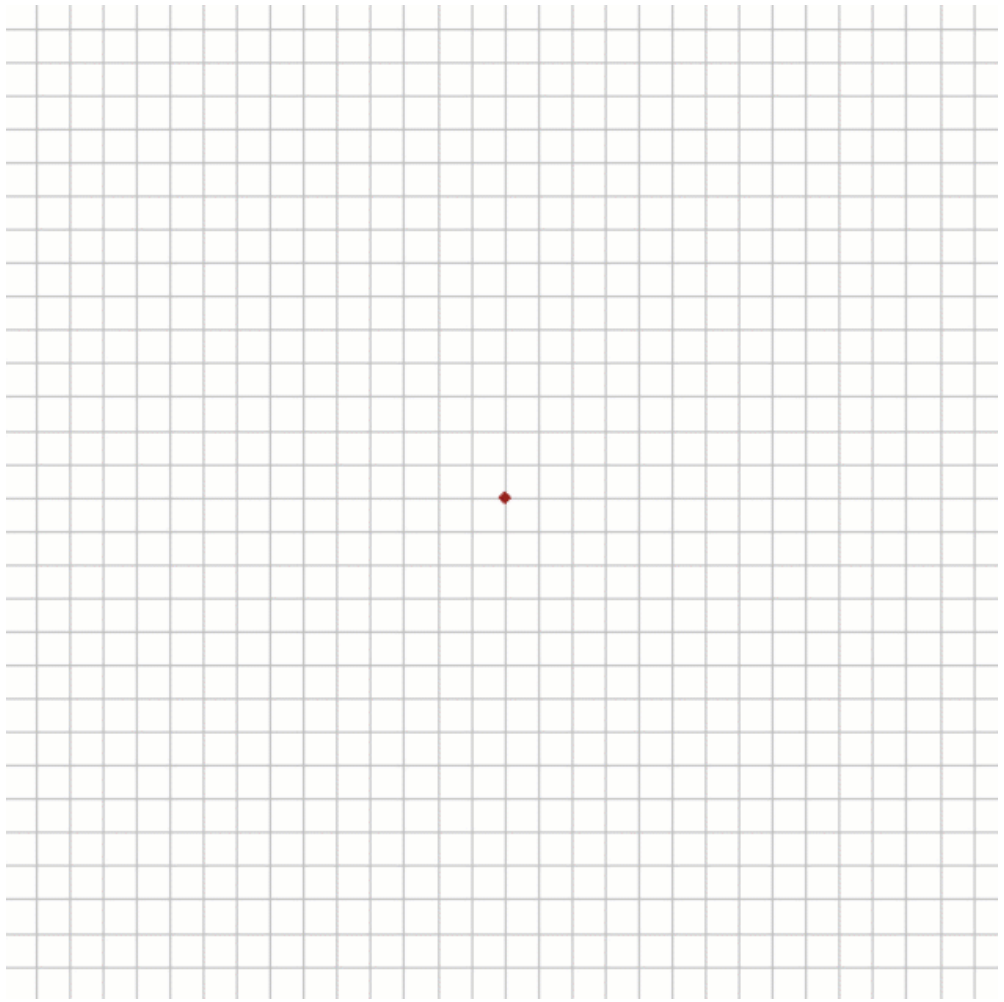
- از مرکز شروع کن.
- به صورت تصادفی به یک تقاطع همسایه برو ولی از هیچ تقاطعی دو بار گذر نکن.
- پیامد ۱: (فرار) رسیدن به لبه صفحه مشبک.
- پیامد ۲: (بن‌بست): یک تقاطع بدون همسایه‌های ملاقات نشده.



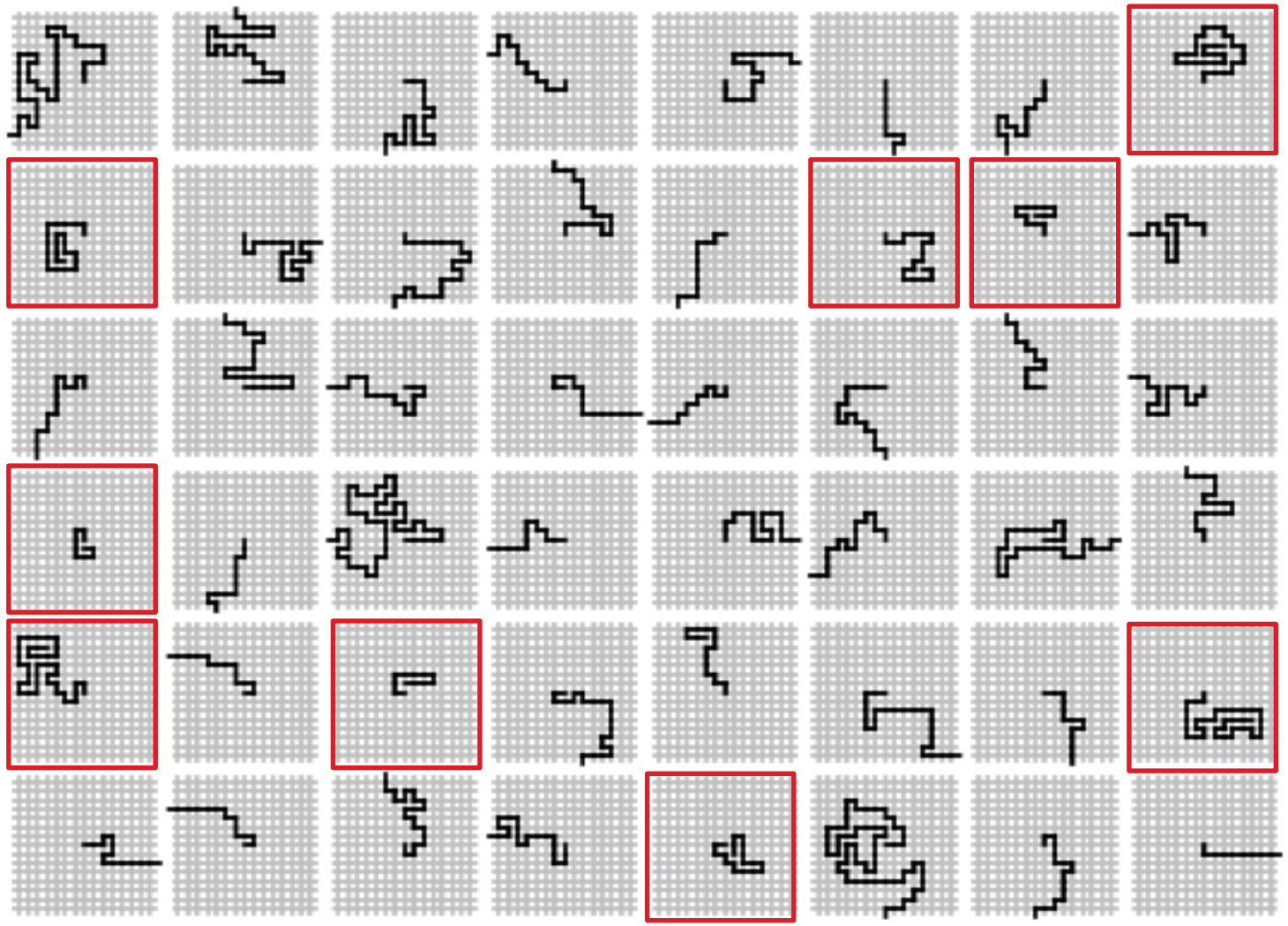
احتمال بن‌بست؟
شبیه‌سازی مونت-کارلو

مسئله ولگشت خودگریز



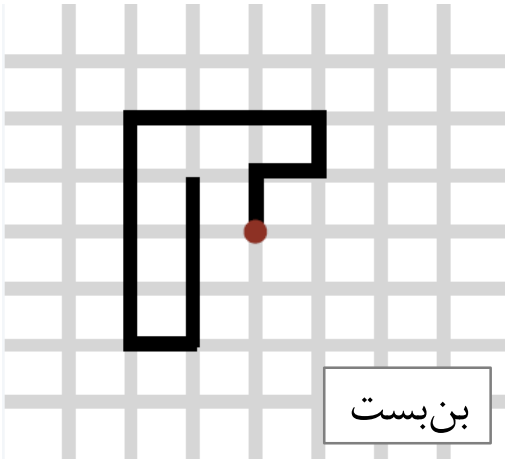


مسئله ولگشت خودگریز



مسئله ولگشت خودگریز

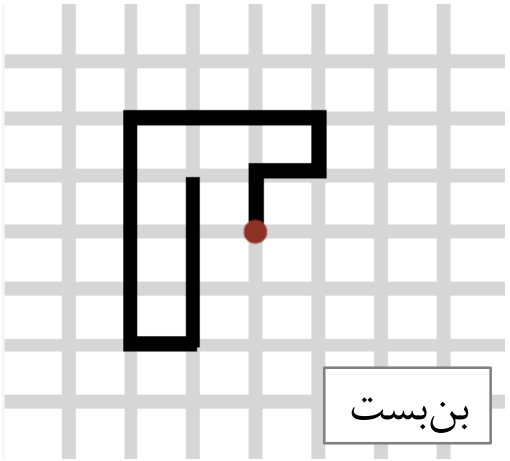

```
import sys, random

n = int(sys.argv[1])
trials = int(sys.argv[2])
dead_ends = 0

for t in range(trials):

    a = [[False] * n for i in range(n)]
    x, y = n // 2, n // 2

    while 0 < x < n - 1 and 0 < y < n - 1:
        if a[x-1][y] and a[x+1][y] and a[x][y-1] and a[x][y+1]:
            dead_ends += 1
            break

        a[x][y] = True

        r = random.random()
        if r < 0.25:
            if not a[x+1][y]: x += 1   # UP
        elif r < 0.5:
            if not a[x-1][y]: x -= 1   # DOWN
        ...
```

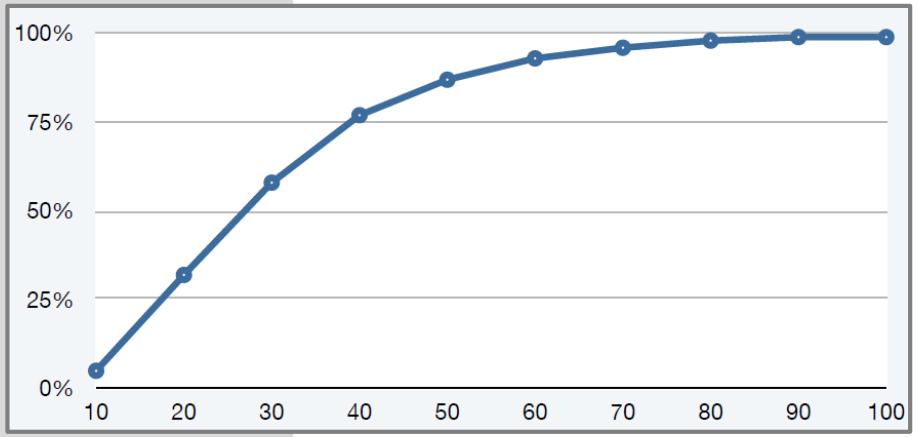



```
% python self_avoiding_random_walk.py 10 100000
5% dead ends

% python self_avoiding_random_walk.py 20 100000
32% dead ends

% python self_avoiding_random_walk.py 30 100000
58% dead ends
```

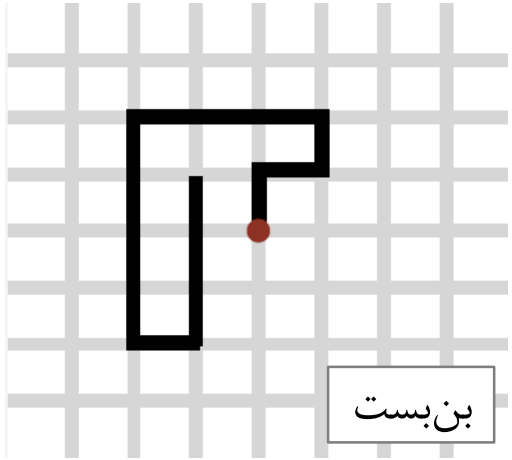
مسئله ولگشت خودگریز


```
import sys, random

n = int(sys.argv[1])
trials = int(sys.argv[2])
dead_ends = 0

for t in range(trials):

    a = [[False] * n for i in range(n)]
    x, y = n // 2, n // 2

    while 0 < x < n - 1 and 0 < y < n - 1:
        if a[x-1][y] and a[x+1][y] and a[x][y-1] and a[x][y+1]:
            dead_ends += 1
            break

    a[x][y] = True

    r = random.random()
    if r < 0.25:
        if not a[x+1][y]: x += 1   # UP
    elif r < 0.5:
        if not a[x-1][y]: x -= 1   # DOWN
    ...
```





شبیه‌سازی، تصادفی بودن و تحلیل

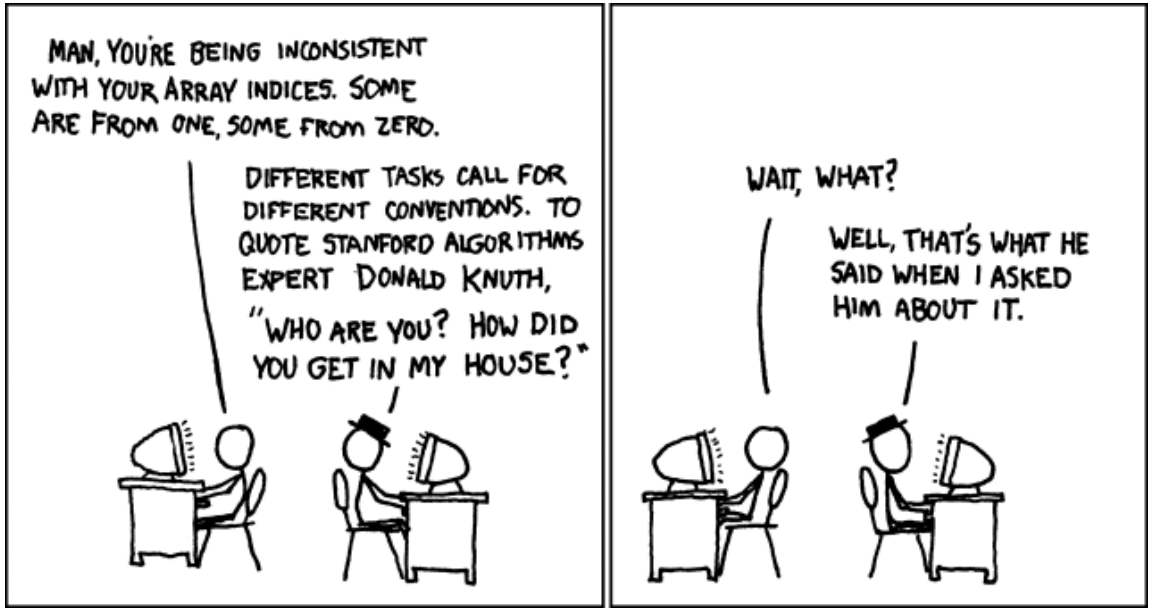


پل فلوری
جایزه نوبل: ۱۹۷۴

- ولگشت خودگریز در یک صفحه مشبک n در n.
- کاربردها.
  - مدل‌سازی رفتار حلال‌ها و بَسپارها (پلیمرها)
  - مدل‌سازی فیزیک مواد مغناطیسی.
  - و مدل‌سازی بسیاری از پدیده‌های فیزیکی دیگر.
- پرسش. احتمال رسیدن به بن‌بست چقدر است؟
  - پاسخ. هیچ کس نمی‌داند (به رغم ده‌ها سال مطالعه). ← *ریاضی‌دان‌ها و پژوهشگران فیزیک نمی‌توانند این مسئله را حل کنند.*
  - پاسخ. بیش از ۹۹ درصد برای $n > 100$. (نتایج شبیه‌سازی). ← *شما می‌توانید!*
- ملاحظه. شبیه‌سازی کامپیوتری در اغلب موارد تنها روش مؤثر برای مطالعه یک پدیده علمی است.

خلاصه

- آرایه‌ها.
  - یک روش سازمان یافته به منظور ذخیره‌سازی حجم انبوهی از داده‌ها.
  - استفاده از آرایه تقریباً به اندازه استفاده از انواع اولیه، ساده است.
  - با داشتن اندیس یک عنصر، می‌توانیم به صورت مستقیم به آن عنصر دستیابی داشته باشیم.



- گام بعدی.
  - خواندن مقدار زیادی داده از فایل و
  - ذخیره‌سازی آنها در یک آرایه.