Lecture 20: Codes & Codebreakers

Comp 102

Fakhir Shaheen

Forman Christian University

A Locked Letter

A Locked Letter

WZZR WZ TR RMZ MGFIZ

GK LUIPGW TR IZXZY



A Locked Letter

WZZR WZ TR RMZ MGFIZ

GK LUIPGW TR IZXZY



- Each letter was swapped for another letter.

A Locked Letter

WZZR WZ TR RMZ MGFIZ
GK LUIPGW TR IZXZY



- Each letter was swapped for another letter.
- Ways to swap: **403,291,461,126,605,635,584,000,000**.

A Locked Letter

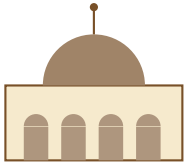
WZZR WZ TR RMZ MGFIZ
GK LUIPGW TR IZXZY



- Each letter was swapped for another letter.
- Ways to swap: **403,291,461,126,605,635,584,000,000**.

No key. No computer. Can a person read it?
Write **YES** *or* **NO** *on your paper (30 s).*

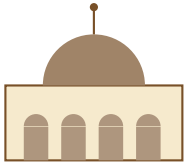
Baghdad, About 850 CE



House of Wisdom



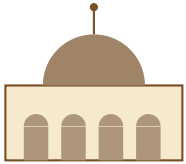
Baghdad, About 850 CE



House of Wisdom



Baghdad, About 850 CE

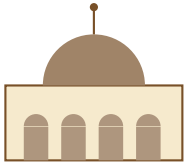


House of Wisdom

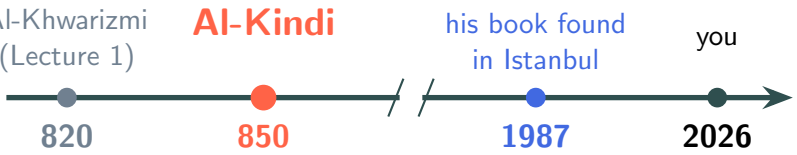


- Al-Kindi wrote the **first** known book on **breaking** codes.

Baghdad, About 850 CE

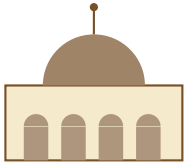


House of Wisdom

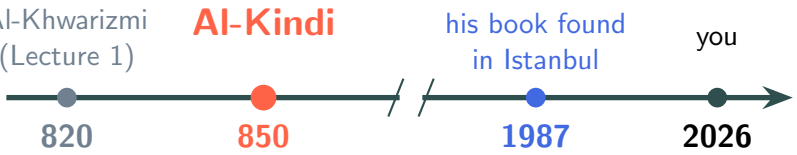


- Al-Kindi wrote the **first** known book on **breaking** codes.

Baghdad, About 850 CE



House of Wisdom



- Al-Kindi wrote the **first** known book on **breaking** codes.
- His answer to our question: **YES**. Today we see how.

Today: A Contest





1	?	?
2	?	?
3	?	?
now	?	?

Today: A Contest



Codemaker



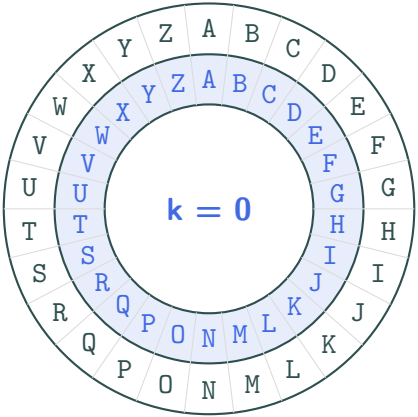
Codebreaker

1	?	?
2	?	?
3	?	?
now	?	?

The **codemaker** locks. The **codebreaker** reads it with **no key**.

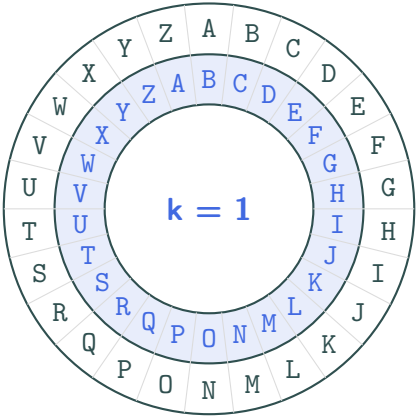
Round 1: The Shift

Caesar's Wheel



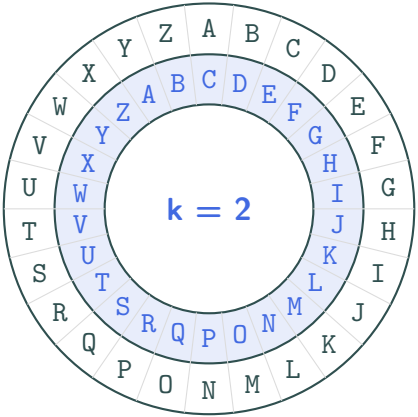
- Julius Caesar, 2,000 years ago.

Caesar's Wheel



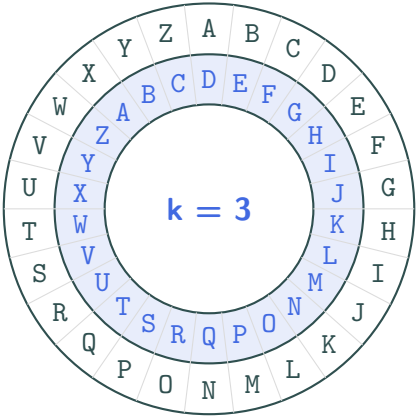
- Julius Caesar, 2,000 years ago.
- Turn the **inner** ring.

Caesar's Wheel



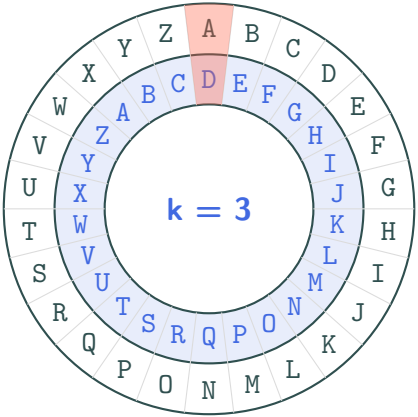
- Julius Caesar, 2,000 years ago.
- Turn the **inner** ring.

Caesar's Wheel



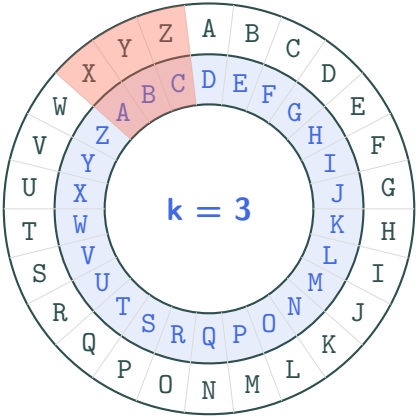
- Julius Caesar, 2,000 years ago.
- Turn the **inner** ring.
- The **key**: how far. Here, **3**.

Caesar's Wheel



- Julius Caesar, 2,000 years ago.
- Turn the **inner** ring.
- The **key**: how far. Here, **3**.
- A → D, B → E, ...

Caesar's Wheel



- Julius Caesar, 2,000 years ago.
- Turn the **inner** ring.
- The **key**: how far. Here, **3**.
- A → D, B → E, ...
- Past Z? Go round: X → A

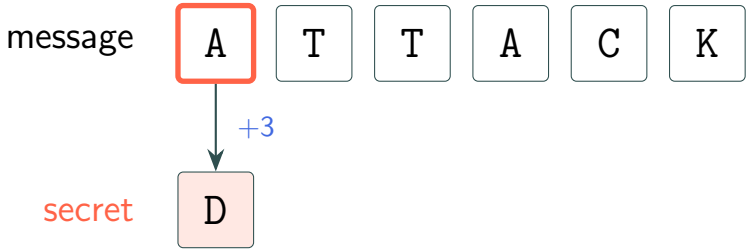
Lock a Word

message

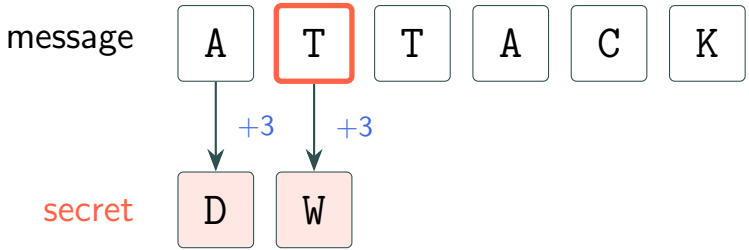


secret

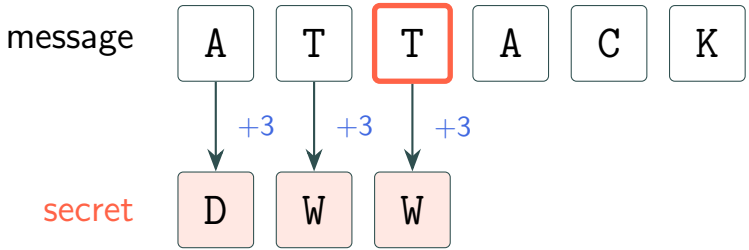
Lock a Word



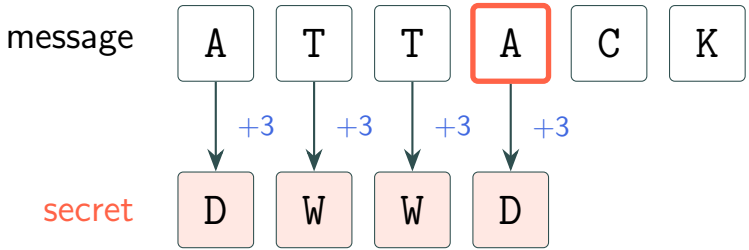
Lock a Word



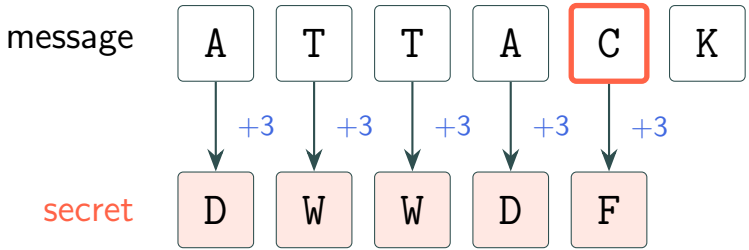
Lock a Word



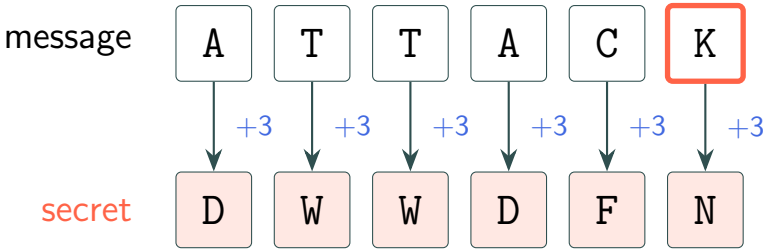
Lock a Word



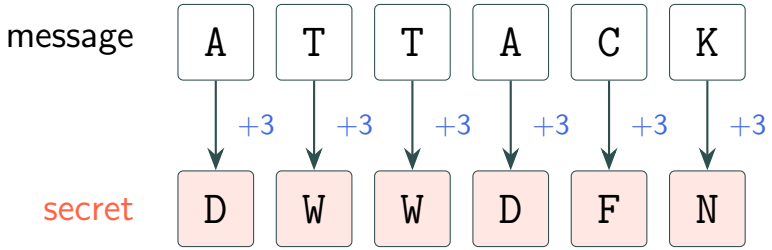
Lock a Word



Lock a Word



Lock a Word



To **unlock**: move each letter **back** 3.

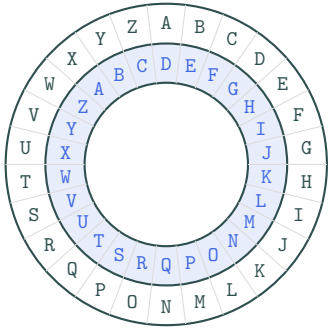
You Try! *(write each answer, 1 min)*

Key = **3**.

(a) Lock CAT.

(b) Unlock KHOOR.

(c) Lock ZOO.



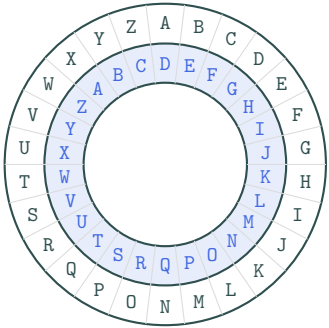
You Try! *(write each answer, 1 min)*

Key = **3**.

(a) Lock `CAT`. `FDW`

(b) Unlock `KHOOR`.

(c) Lock `ZOO`.



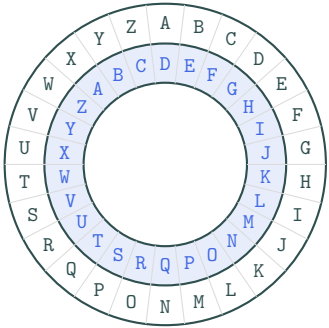
You Try! *(write each answer, 1 min)*

Key = **3**.

(a) Lock `CAT`. `FDW`

(b) Unlock `KHOOR`. `HELLO`

(c) Lock `ZOO`.



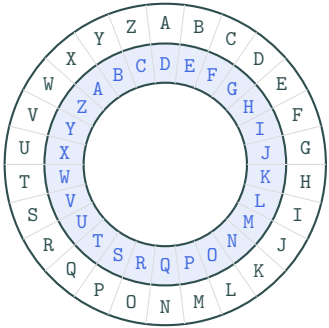
You Try! *(write each answer, 1 min)*

Key = **3**.

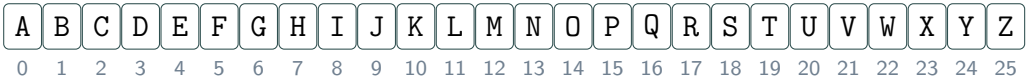
(a) Lock `CAT`. `FDW`

(b) Unlock `KHOOR`. `HELLO`

(c) Lock `ZOO`. `CRR`

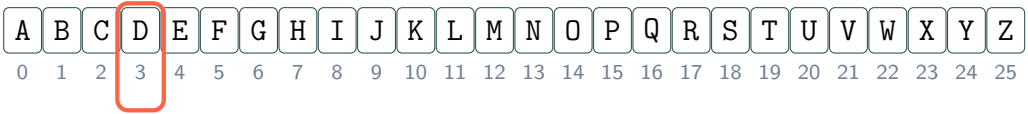


Letters Are Places



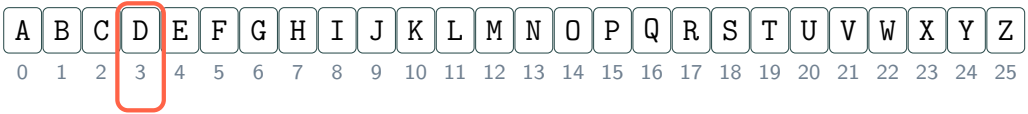

```
ABC = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
ABC.index('D')
ABC[3]
ABC[3 + 3]
```

Letters Are Places



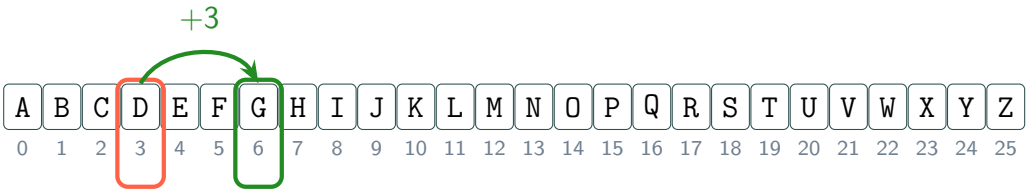

```
ABC = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
ABC.index('D')        # 3
ABC[3]
ABC[3 + 3]
```

Letters Are Places



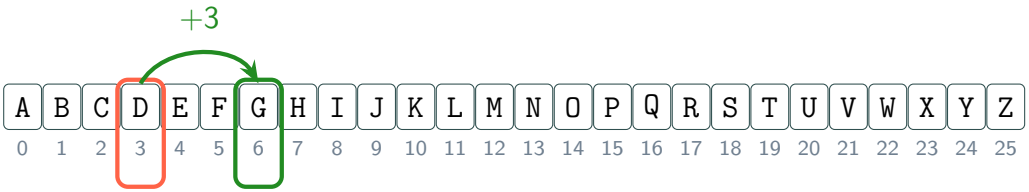

```
ABC = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
ABC.index('D')        # 3
ABC[3]                # 'D'
ABC[3 + 3]
```

Letters Are Places




```
ABC = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
ABC.index('D')          # 3
ABC[3]                  # 'D'
ABC[3 + 3]              # 'G'
```

Letters Are Places




```
ABC = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
ABC.index('D')          # 3
ABC[3]                  # 'D'
ABC[3 + 3]              # 'G'
```

Lock a letter: find its **place**, add the **key**, read the new letter.

Bug Hunt! *(write the line number and the fix, 1 min)*

A first try at Caesar's code:

```
1  def shift(msg, k):
2      out = ''
3      for ch in msg:
4          if ch in ABC:
5              i = ABC.index(ch)
6              out += ABC[i + k]
7          else:
8              out += ch
9      return out
10
11 print(shift('HELLO', 3))
12 print(shift('ZOO', 3))
```

Symptom:

```
KHOOR
IndexError: string
index out of range
```

Bug Hunt! *(write the line number and the fix, 1 min)*

A first try at Caesar's code:

```
1  def shift(msg, k):
2      out = ''
3      for ch in msg:
4          if ch in ABC:
5              i = ABC.index(ch)
6              out += ABC[i + k]
7          else:
8              out += ch
9      return out
10
11 print(shift('HELLO', 3))
12 print(shift('ZOO', 3))
```

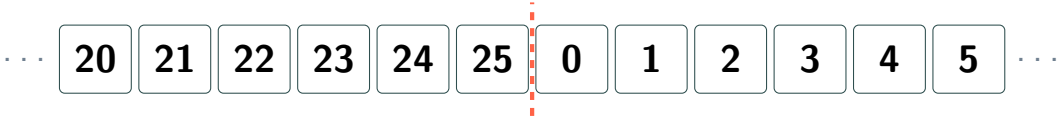
Symptom:

```
KHOOR
IndexError: string
index out of range
```

Line 6. `Z` is place 25. 25 + 3 = 28, and `ABC` has no place 28. The code forgot to **go round** the wheel.

Fix: `ABC[(i + k) % 26]`

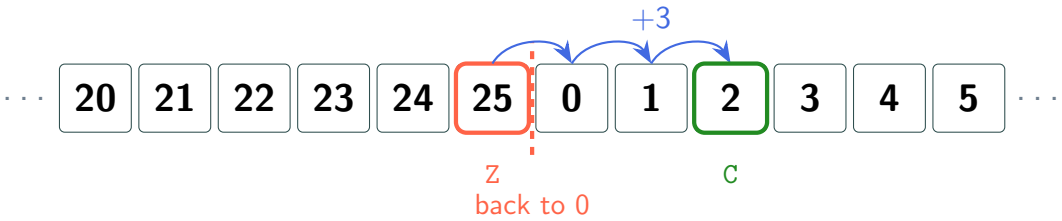
% Goes Round the Wheel




```
(25 + 3) % 26
```

You Try! *(predict)* `26 % 26` `-1 % 26`

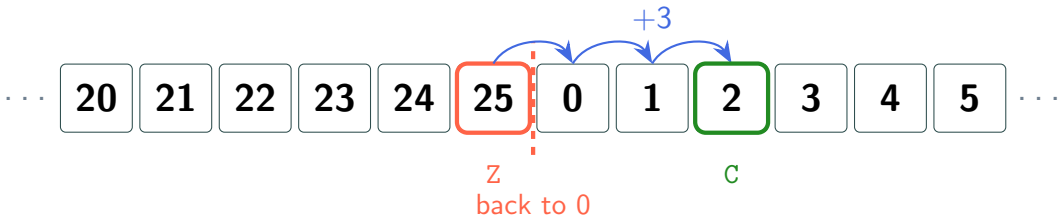
% Goes Round the Wheel




```
(25 + 3) % 26
```

You Try! *(predict)* `26 % 26` `-1 % 26`

% Goes Round the Wheel



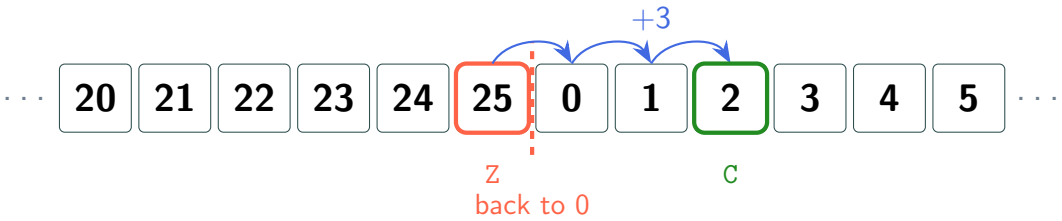

```
(25 + 3) % 26      # 2
```

You Try! *(predict)* `26 % 26`

The **remainder** after dividing by 26 (Lecture 2).

`-1 % 26`

% Goes Round the Wheel

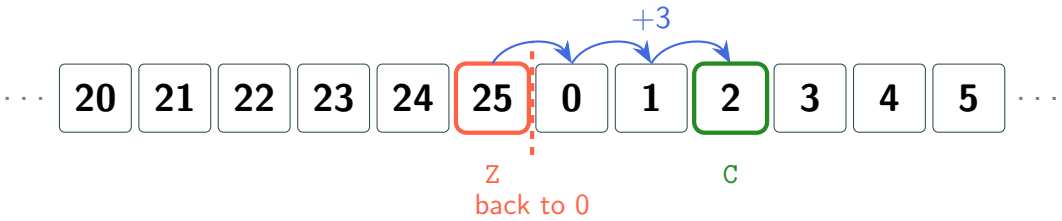



```
(25 + 3) % 26      # 2
```

The **remainder** after dividing by 26 (Lecture 2).

You Try! *(predict)* 26 % 26 → **0** −1 % 26

% Goes Round the Wheel




```
(25 + 3) % 26     # 2
```

The **remainder** after dividing by 26 (Lecture 2).

You Try! *(predict)* 26 % 26 → **0**

−1 % 26 → **25**, so `A` back 1 is `Z`.

Caesar's Code in Python

```
1  def shift(msg, k):
2      """Moves each letter k places."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              i = ABC.index(ch)
7              out += ABC[(i + k) % 26]
8          else:
9              out += ch
10     return out
```

Caesar's Code in Python

```
1  def shift(msg, k):
2      """Moves each letter k places."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              i = ABC.index(ch)
7              out += ABC[(i + k) % 26]
8          else:
9              out += ch
10     return out
```

- **Line 5:** only letters move.

Caesar's Code in Python

```
1  def shift(msg, k):
2      """Moves each letter k places."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              i = ABC.index(ch)
7              out += ABC[(i + k) % 26]
8          else:
9              out += ch
10     return out
```

- **Line 5:** only letters move.
- **Line 9:** a space stays a space.

Caesar's Code in Python

```
1  def shift(msg, k):
2      """Moves each letter k places."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              i = ABC.index(ch)
7              out += ABC[(i + k) % 26]
8          else:
9              out += ch
10     return out

   shift('ATTACK', 3)
   shift('DWWDFN', -3)
```

- **Line 5:** only letters move.
- **Line 9:** a space stays a space.

Caesar's Code in Python

```
1  def shift(msg, k):
2      """Moves each letter k places."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              i = ABC.index(ch)
7              out += ABC[(i + k) % 26]
8          else:
9              out += ch
10     return out

   shift('ATTACK', 3)       # 'DWWDFN'
   shift('DWWDFN', -3)
```

- **Line 5:** only letters move.
- **Line 9:** a space stays a space.

Caesar's Code in Python

```
1  def shift(msg, k):
2      """Moves each letter k places."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              i = ABC.index(ch)
7              out += ABC[(i + k) % 26]
8          else:
9              out += ch
10     return out

   shift('ATTACK', 3)       # 'DWWDFN'
   shift('DWWDFN', -3)      # 'ATTACK'
```

- **Line 5:** only letters move.
- **Line 9:** a space stays a space.

Caesar's Code in Python

```
1  def shift(msg, k):
2      """Moves each letter k places."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              i = ABC.index(ch)
7              out += ABC[(i + k) % 26]
8          else:
9              out += ch
10     return out
```

```
shift('ATTACK', 3)      # 'DWWDFN'
shift('DWWDFN', -3)     # 'ATTACK'
```

Key `k` locks. Key `-k` unlocks.

- **Line 5:** only letters move.
- **Line 9:** a space stays a space.

Break It: Try Every Key

A spy has the secret, but **not** the key.

```
secret = 'WKH NHB LV XQGHU WKH PDW'
for k in range(26):
    print(k, shift(secret, -k))
```

Break It: Try Every Key

A spy has the secret, but **not** the key.

```
secret = 'WKH NHB LV XQGHU WKH PDW'
for k in range(26):
    print(k, shift(secret, -k))
```

```
 0  WKH NHB LV XQGHU WKH PDW
 1  VJG MGA KU WPFGT VJG OCV
 2  UIF LFZ JT VOEFS UIF NBU
 3  THE KEY IS UNDER THE MAT
 4  SGD JDX HR TMCDQ SGD LZS
 5  RFC ICW GQ SLBCP RFC KYR
 6  QEB HBV FP RKABO QEB JXQ
 7  PDA GAU EO QJZAN PDA IWP
 8  OCZ FZT DN PIYZM OCZ HVO
 9  NBY EYS CM OHXYL NBY GUN
10  MAX DXR BL NGWXK MAX FTM
...
24  YMJ PJD NX ZSIJW YMJ RFY
25  XLI OIC MW YRHIV XLI QEX
```

Break It: Try Every Key

A spy has the secret, but **not** the key.

```
secret = 'WKH NHB LV XQGHU WKH PDW'
for k in range(26):
    print(k, shift(secret, -k))
```

Only **26** keys. Done before you blink.

```
 0  WKH NHB LV XQGHU WKH PDW
 1  VJG MGA KU WPFGT VJG OCV
 2  UIF LFZ JT VOEFS UIF NBU
 3  THE KEY IS UNDER THE MAT
 4  SGD JDX HR TMCDQ SGD LZS
 5  RFC ICW GQ SLBCP RFC KYR
 6  QEB HBV FP RKABO QEB JXQ
 7  PDA GAU EO QJZAN PDA IWP
 8  OCZ FZT DN PIYZM OCZ HVO
 9  NBY EYS CM OHXYL NBY GUN
10  MAX DXR BL NGWXK MAX FTM
...
24  YMJ PJD NX ZSIJW YMJ RFY
25  XLI OIC MW YRHIV XLI QEX
```

Break It: Try Every Key

A spy has the secret, but **not** the key.

```
secret = 'WKH NHB LV XQGHU WKH PDW'
for k in range(26):
    print(k, shift(secret, -k))
```

Only **26** keys. Done before you blink.

Like the PIN game (Lecture 15): **brute force**.

```
 0  WKH NHB LV XQGHU WKH PDW
 1  VJG MGA KU WPFGT VJG OCV
 2  UIF LFZ JT VOEFS UIF NBU
 3  THE KEY IS UNDER THE MAT
 4  SGD JDX HR TMCDQ SGD LZS
 5  RFC ICW GQ SLBCP RFC KYR
 6  QEB HBV FP RKABO QEB JXQ
 7  PDA GAU EO QJZAN PDA IWP
 8  OCZ FZT DN PIYZM OCZ HVO
 9  NBY EYS CM OHXYL NBY GUN
10  MAX DXR BL NGWXK MAX FTM
...
24  YMJ PJD NX ZSIJW YMJ RFY
25  XLI OIC MW YRHIV XLI QEX
```

Round 1



Codemaker



Codebreaker

1	Shift every letter	Try all 26 keys ✓
2	?	?
3	?	?
now	?	?

Round 2: The Mix

Mix the Whole Alphabet

letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

becomes

A	T	T	A	C	K

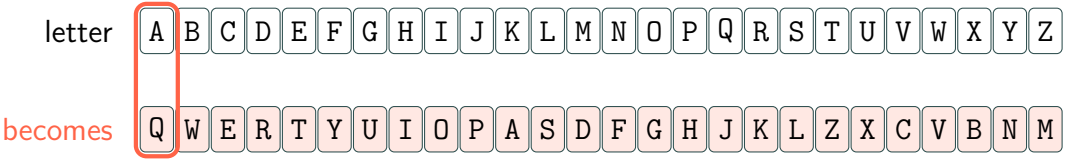
Mix the Whole Alphabet

letter	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
becomes	Q	W	E	R	T	Y	U	I	O	P	A	S	D	F	G	H	J	K	L	Z	X	C	V	B	N	M

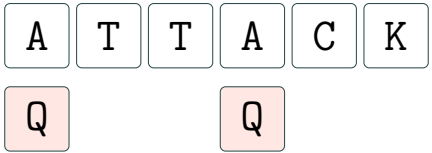
The key: 26 letters, **mixed**.

A	T	T	A	C	K

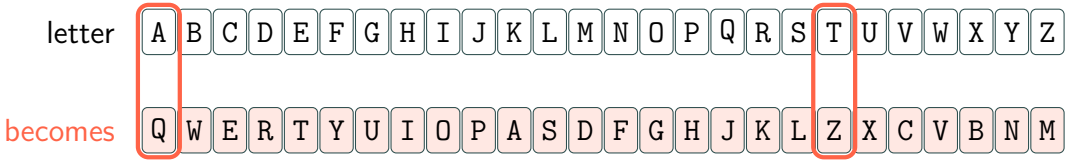
Mix the Whole Alphabet



The key: 26 letters, **mixed**.



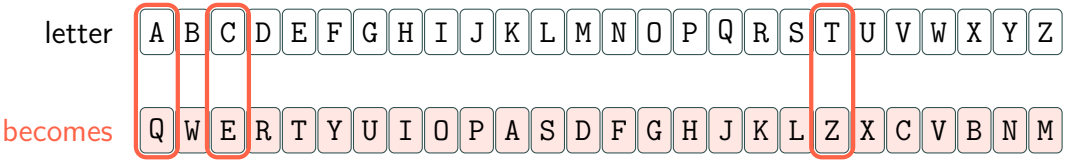
Mix the Whole Alphabet



The key: 26 letters, **mixed**.

A	T	T	A	C	K
Q	Z	Z	Q		

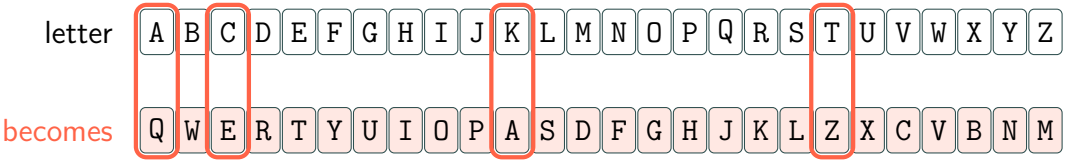
Mix the Whole Alphabet



The key: 26 letters, **mixed**.

A	T	T	A	C	K
Q	Z	Z	Q	E	

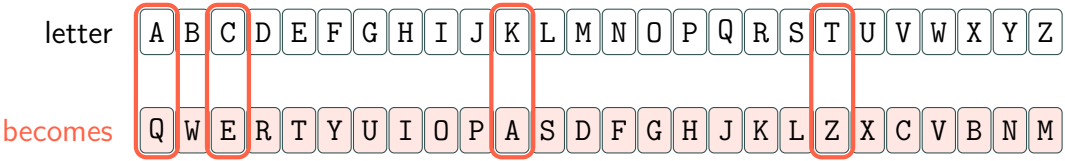
Mix the Whole Alphabet



The key: 26 letters, **mixed**.

A	T	T	A	C	K
Q	Z	Z	Q	E	A

Mix the Whole Alphabet



The key: 26 letters, **mixed**.

Here: the keyboard, QWERTY...

A	T	T	A	C	K
Q	Z	Z	Q	E	A

Mixing in Python

```
1  def substitute(msg, key):
2      """key: the 26 letters, mixed."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              out += key[ABC.index(ch)]
7          else:
8              out += ch
9      return out
```

Mixing in Python

```
1  def substitute(msg, key):
2      """key: the 26 letters, mixed."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              out += key[ABC.index(ch)]
7          else:
8              out += ch
9      return out

   KEY = 'QWERTYUIOPASDFGHJKLZXCVBNM'
   substitute('ATTACK', KEY)
```

Mixing in Python

```
1  def substitute(msg, key):
2      """key: the 26 letters, mixed."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              out += key[ABC.index(ch)]
7          else:
8              out += ch
9      return out

   KEY = 'QWERTYUIOPASDFGHJKLZXCVBNM'
   substitute('ATTACK', KEY)     # 'QZZQEA'
```

Mixing in Python

```
1  def substitute(msg, key):
2      """key: the 26 letters, mixed."""
3      out = ''
4      for ch in msg:
5          if ch in ABC:
6              out += key[ABC.index(ch)]
7          else:
8              out += ch
9      return out

   KEY = 'QWERTYUIOPASDFGHJKLZXCVBNM'
   substitute('ATTACK', KEY)     # 'QZZQEA'
```

Line 6: the place of `ch` in `ABC` → the letter at that place in `key`.

You Try! *(read and explain, 2 min)*

```
1  def mystery(secret, key):
2      out = ''
3      for ch in secret:
4          if ch in key:
5              out += ABC[key.index(ch)]
6          else:
7              out += ch
8      return out
```

```
ABC = ABCDEFGHIJKLMNOPQRSTUVWXYZ
KEY = QWERTYUIOPASDFGHJKLZXCVBNM
```

(a) `mystery('QZZQEA', KEY)`

(b) In **one** sentence: what does `mystery` do?

You Try! *(read and explain, 2 min)*

```
1  def mystery(secret, key):
2      out = ''
3      for ch in secret:
4          if ch in key:
5              out += ABC[key.index(ch)]
6          else:
7              out += ch
8      return out
```

```
ABC = ABCDEFGHIJKLMNOPQRSTUVWXYZ
KEY = QWERTYUIOPASDFGHJKLZXCVBNM
```

(a) `mystery('QZZQEA', KEY)`

`'ATTACK'`

(b) In **one** sentence: what does `mystery` do?

You Try! *(read and explain, 2 min)*

```
1  def mystery(secret, key):
2      out = ''
3      for ch in secret:
4          if ch in key:
5              out += ABC[key.index(ch)]
6          else:
7              out += ch
8      return out
```

```
ABC = ABCDEFGHIJKLMNOPQRSTUVWXYZ
KEY = QWERTYUIOPASDFGHJKLZXCVBNM
```

(a) `mystery('QZZQEA', KEY)`

`'ATTACK'`

(b) In **one** sentence: what does `mystery` do?

It **unlocks**: it undoes substitute.

You Try! *(read and explain, 2 min)*

```
1  def mystery(secret, key):
2      out = ''
3      for ch in secret:
4          if ch in key:
5              out += ABC[key.index(ch)]
6          else:
7              out += ch
8      return out
```

```
ABC = ABCDEFGHIJKLMNOPQRSTUVWXYZ
KEY = QWERTYUIOPASDFGHJKLZXCVBNM
```

(a) `mystery('QZZQEA', KEY)`

`'ATTACK'`

(b) In **one** sentence: what does `mystery` do?

It **unlocks**: it undoes substitute.

Line 5 goes the other way: the place of `ch` in `key` → the letter at that place in `ABC`.

How Many Keys?



How Many Keys?

26 letters to choose from



26

How Many Keys?

one is used: 25 left



26×25

How Many Keys?



$26 \times 25 \times 24$

How Many Keys?



$$26 \times 25 \times 24 \times 23 \times 22 \times 21 \times \ldots \times 1$$

How Many Keys?



$$26 \times 25 \times 24 \times 23 \times 22 \times 21 \times \ldots \times 1$$

$= 26!$ (Lecture 19: n items have $n!$ orders)

How Many Keys?



$$26 \times 25 \times 24 \times 23 \times 22 \times 21 \times \ldots \times 1$$

$= 26!$ (Lecture 19: n items have $n!$ orders)

403,291,461,126,605,635,584,000,000 keys

You Try! *(write your answer, 2 min)*

A computer tries **10 million** keys each second. How long to try all 26!?

Use: $26! \approx 4 \times 10^{26}$ 1 year $\approx 3 \times 10^7$ seconds

(a) Seconds?

(b) Years?

You Try! *(write your answer, 2 min)*

A computer tries **10 million** keys each second. How long to try all 26!?

Use: $26! \approx 4 \times 10^{26}$ $1 \text{ year} \approx 3 \times 10^7 \text{ seconds}$

(a) Seconds?

$4 \times 10^{26} \div 10^7 = 4 \times 10^{19}$ seconds

(b) Years?

You Try! *(write your answer, 2 min)*

A computer tries **10 million** keys each second. How long to try all 26!?

Use: $26! \approx 4 \times 10^{26}$ 1 year $\approx 3 \times 10^7$ seconds

(a) Seconds?

$4 \times 10^{26} \div 10^7 = 4 \times 10^{19}$ seconds

(b) Years?

$4 \times 10^{19} \div 3 \times 10^7 \approx$ **1.3 trillion years**

The Wall



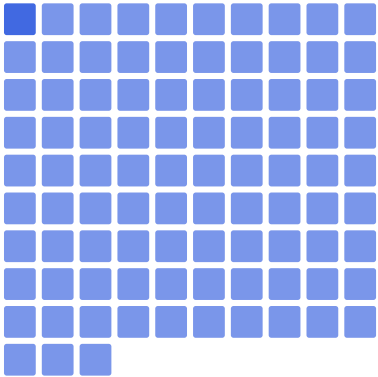
- One square: the **age of the universe**, 13.8 billion years.

The Wall



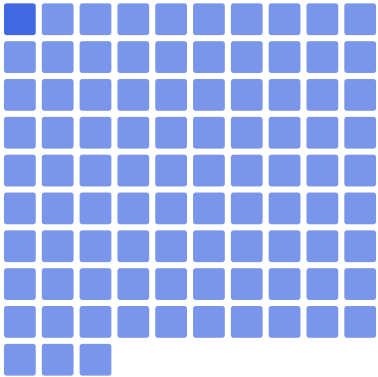
- One square: the **age of the universe**, 13.8 billion years.
- Trying every key: 1.3 trillion years.

The Wall



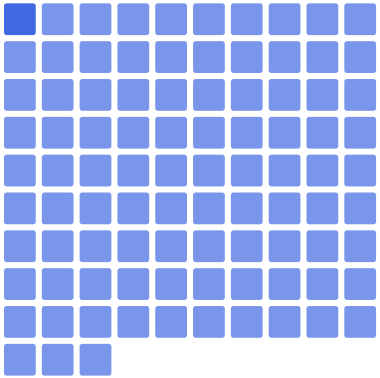
- One square: the **age of the universe**, 13.8 billion years.
- Trying every key: 1.3 trillion years.
- That is **93** squares.

The Wall



- One square: the **age of the universe**, 13.8 billion years.
- Trying every key: 1.3 trillion years.
- That is **93** squares.
- 1,000 computers? Still 1.3 **billion** years.

The Wall



- One square: the **age of the universe**, 13.8 billion years.
- Trying every key: 1.3 trillion years.
- That is **93** squares.
- 1,000 computers? Still 1.3 **billion** years.

So many keys! People were sure: **no one can break it**.

Al-Kindi Counts

Al-Kindi's Idea

From his book, in simple words

1



Take a long letter.
Count each letter.

Al-Kindi's Idea

From his book, in simple words

1



Take a long letter.
Count each letter.

2



Count each letter
of the **secret**.

Al-Kindi's Idea

From his book, in simple words

1



Take a long letter.
Count each letter.

2



Count each letter
of the **secret**.

3

1st ↔ **1st**

2nd ↔ **2nd**

Most common
goes with most common.

Al-Kindi's Idea

From his book, in simple words

1



Take a long letter.
Count each letter.

2



Count each letter
of the **secret**.

3

1st ↔ **1st**

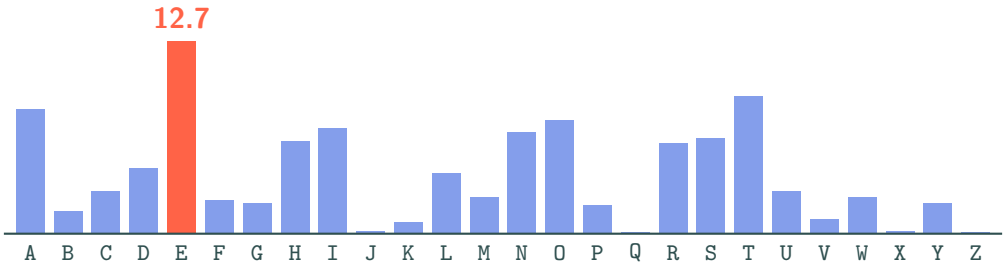
2nd ↔ **2nd**

Most common
goes with most common.

He never tried a single key.

Some Letters Are Common

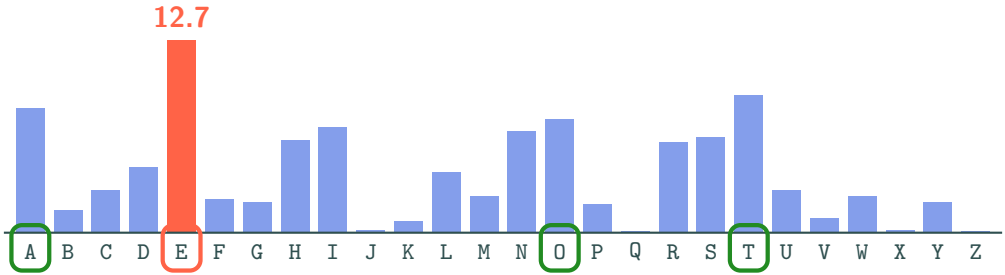
100 letters of English text:



- About **1 letter in 8** is **E**.

Some Letters Are Common

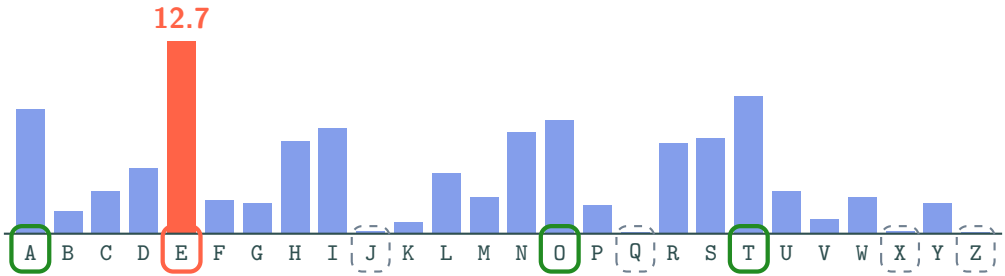
100 letters of English text:



- About **1 letter in 8** is **E**.
- Then **T**, **A**, **O**, ...

Some Letters Are Common

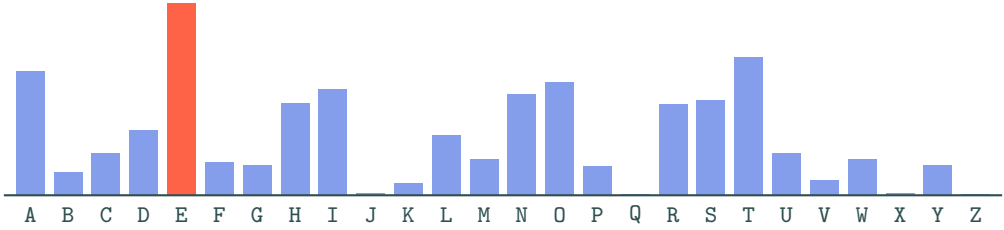
100 letters of English text:



- About **1 letter in 8** is **E**.
- Then **T**, **A**, **O**, . . . **J**, **Q**, **X**, **Z**: almost never.

Mixing Moves the Bars

A long English text:



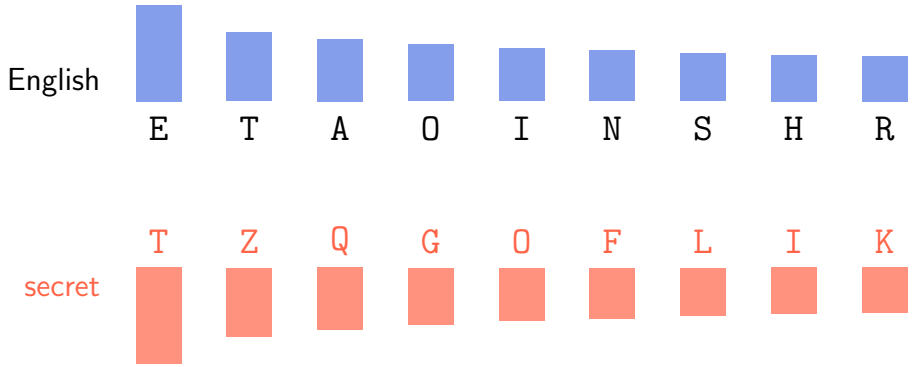
Mixing Moves the Bars

The same text, locked with the `QWERTY` key:



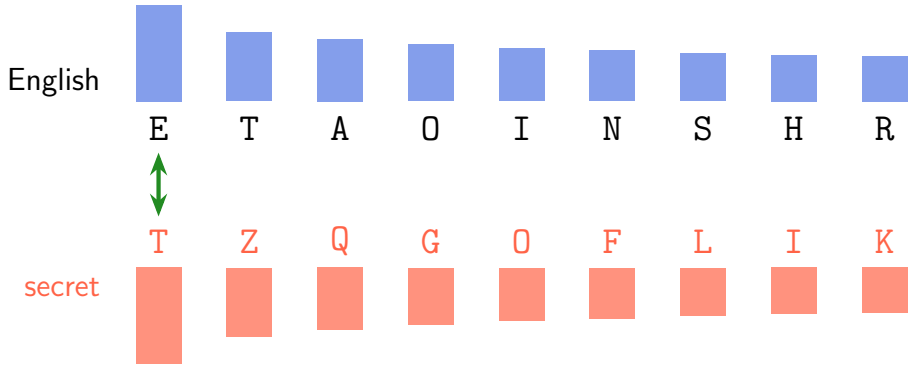
Mixing Moves the Bars

Sort both by height (Lecture 19). Then match.



Mixing Moves the Bars

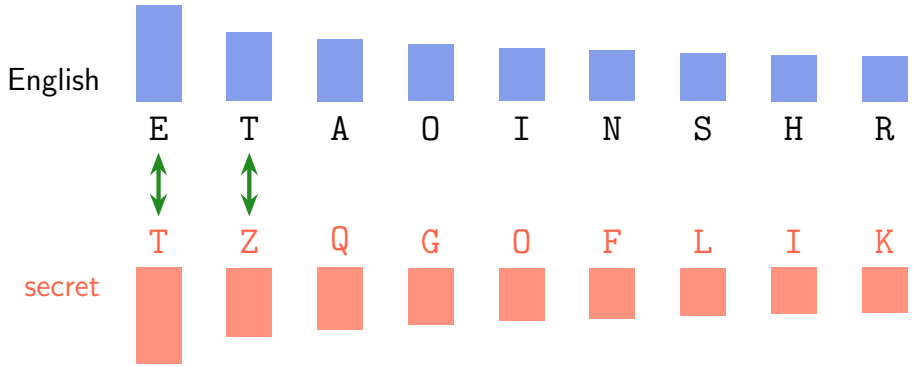
Sort both by height (Lecture 19). Then match.



T stands for E.

Mixing Moves the Bars

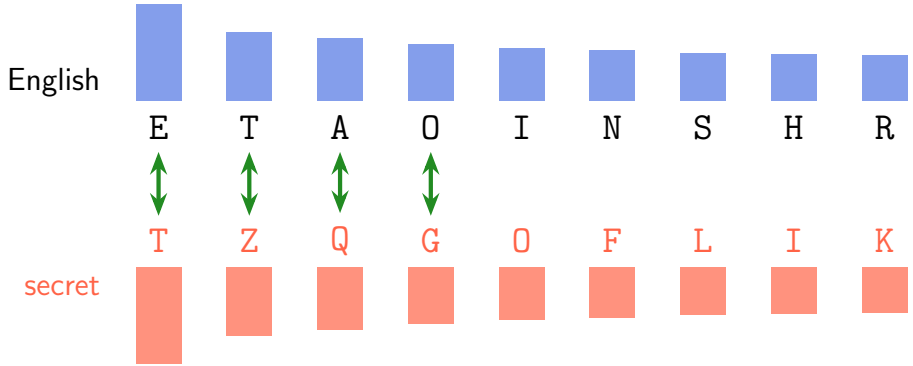
Sort both by height (Lecture 19). Then match.



Z stands for T.

Mixing Moves the Bars

Sort both by height (Lecture 19). Then match.



Mixing hides the **letters**. It does **not** hide the **counts**.

Counting, Step by Step



`counts = {}`

Counting, Step by Step






```
counts = {'S': 1}
```

Counting, Step by Step

S E E T H E T R E E




```
counts = {'S': 1, 'E': 1}
```

Counting, Step by Step

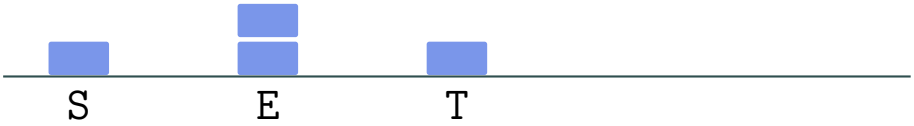
S E E T H E T R E E




```
counts = {'S': 1, 'E': 2}
```

Counting, Step by Step

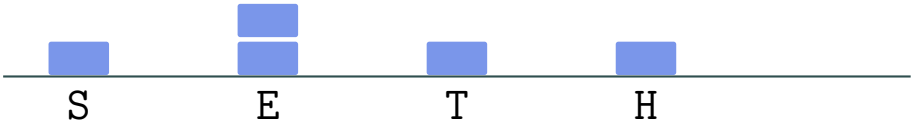





```
counts = {'S': 1, 'E': 2, 'T': 1}
```

Counting, Step by Step

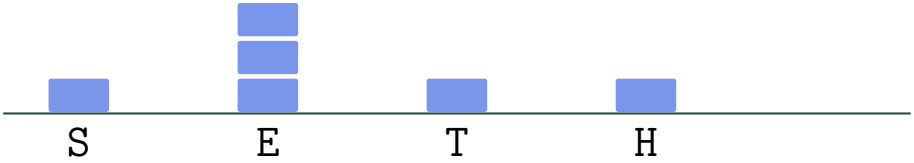





```
counts = {'S': 1, 'E': 2, 'T': 1, 'H': 1}
```

Counting, Step by Step

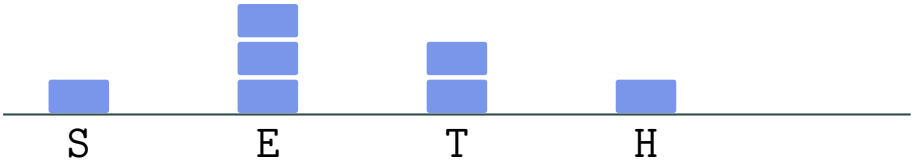





```
counts = {'S': 1, 'E': 3, 'T': 1, 'H': 1}
```

Counting, Step by Step

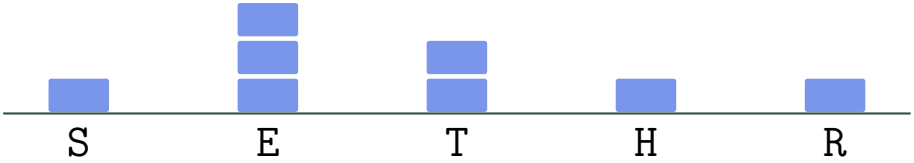





```
counts = {'S': 1, 'E': 3, 'T': 2, 'H': 1}
```

Counting, Step by Step

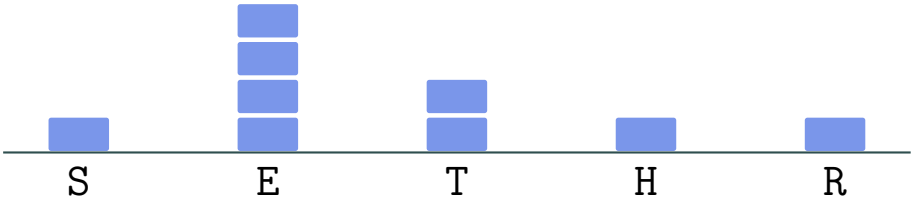





```
counts = {'S': 1, 'E': 3, 'T': 2, 'H': 1, 'R': 1}
```

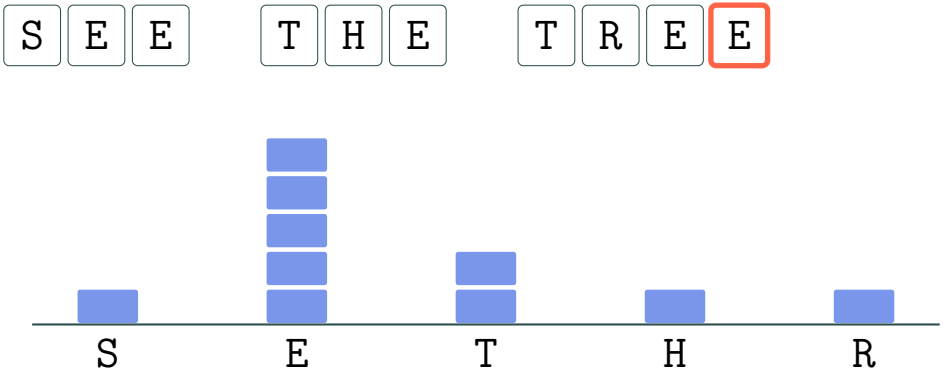
Counting, Step by Step






```
counts = {'S': 1, 'E': 4, 'T': 2, 'H': 1, 'R': 1}
```

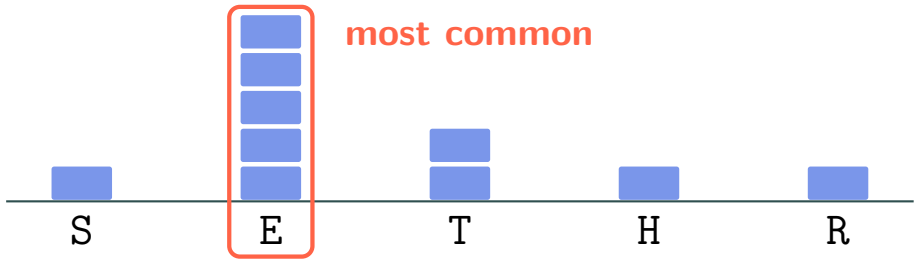
Counting, Step by Step




```
counts = {'S': 1, 'E': 5, 'T': 2, 'H': 1, 'R': 1}
```

Counting, Step by Step

S E E T H E T R E E




```
counts = {'S': 1, 'E': 5, 'T': 2, 'H': 1, 'R': 1}
```

Counting in Python

```
1  def count_letters(text):
2      counts = {}
3      for ch in text:
4          if ch in ABC:
5              if ch in counts:
6                  counts[ch] += 1
7              else:
8                  counts[ch] = 1
9      return counts
```

Lecture 17's word count, for **letters**.

Counting in Python

```
1  def count_letters(text):
2      counts = {}
3      for ch in text:
4          if ch in ABC:
5              if ch in counts:
6                  counts[ch] += 1
7              else:
8                  counts[ch] = 1
9      return counts
```

Lecture 17's word count, for **letters**.

Most common first:

```
def by_count(counts):
    return sorted(counts,
        key=lambda ch: counts[ch],
        reverse=True)
```

Counting in Python

```
1  def count_letters(text):
2      counts = {}
3      for ch in text:
4          if ch in ABC:
5              if ch in counts:
6                  counts[ch] += 1
7              else:
8                  counts[ch] = 1
9      return counts
```

Lecture 17's word count, for **letters**.

Most common first:

```
def by_count(counts):
    return sorted(counts,
        key=lambda ch: counts[ch],
        reverse=True)
```

`sorted(counts)` sorts the **keys**. `key=` sorts them by their counts (Lecture 19).

Counting in Python

```
1  def count_letters(text):
2      counts = {}
3      for ch in text:
4          if ch in ABC:
5              if ch in counts:
6                  counts[ch] += 1
7              else:
8                  counts[ch] = 1
9      return counts
```

Most common first:

```
def by_count(counts):
    return sorted(counts,
        key=lambda ch: counts[ch],
        reverse=True)
```

`sorted(counts)` sorts the **keys**. `key=` sorts them by their counts (Lecture 19).

Lecture 17's word count, for **letters**.

```
by_count(count_letters('SEE THE TREE'))
```

Counting in Python

```
1  def count_letters(text):
2      counts = {}
3      for ch in text:
4          if ch in ABC:
5              if ch in counts:
6                  counts[ch] += 1
7              else:
8                  counts[ch] = 1
9      return counts
```

Most common first:

```
def by_count(counts):
    return sorted(counts,
        key=lambda ch: counts[ch],
        reverse=True)
```

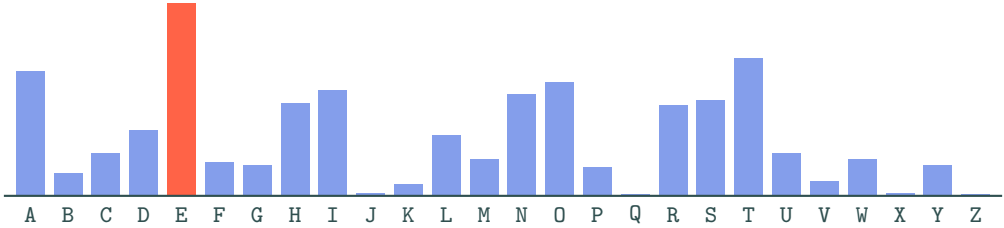
`sorted(counts)` sorts the **keys**. `key=` sorts them by their counts (Lecture 19).

Lecture 17's word count, for **letters**.

```
by_count(count_letters('SEE THE TREE'))
# ['E', 'T', 'S', 'H', 'R']
```

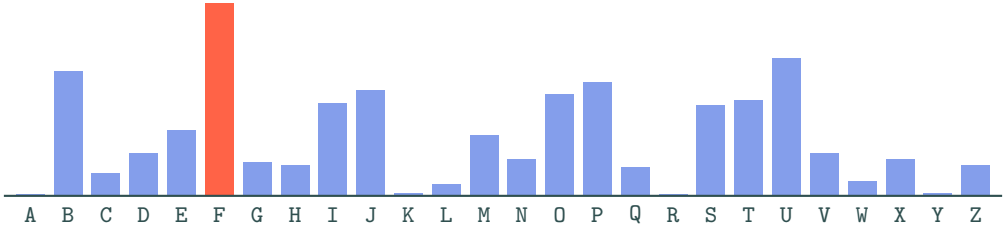
Counting Breaks Caesar Too

English text, **not** locked.



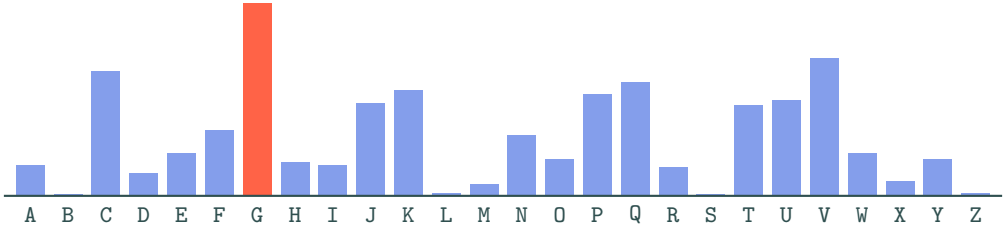
Counting Breaks Caesar Too

Lock it with a shift: every bar moves along.



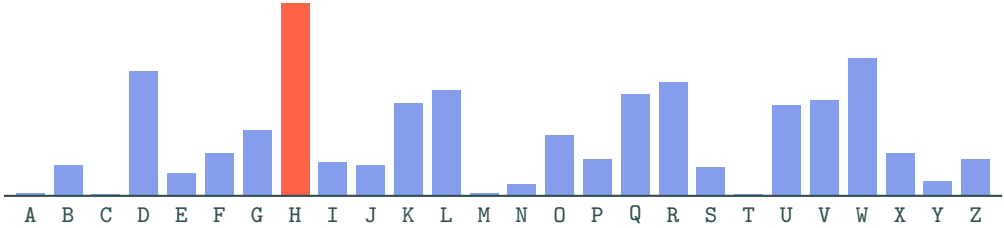
Counting Breaks Caesar Too

Lock it with a shift: every bar moves along.



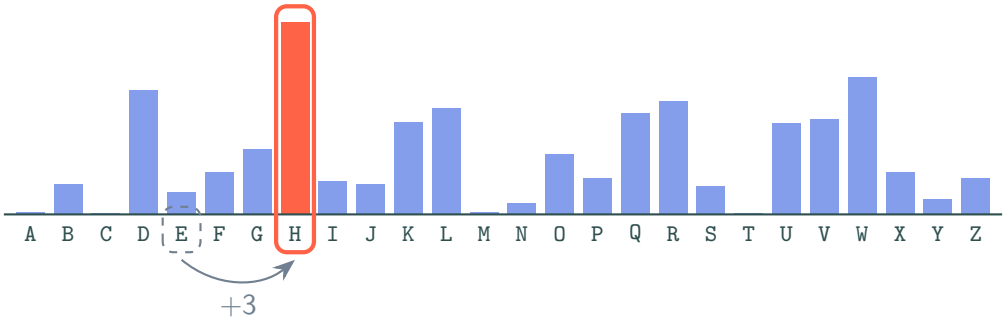
Counting Breaks Caesar Too

Lock it with a shift: every bar moves along.



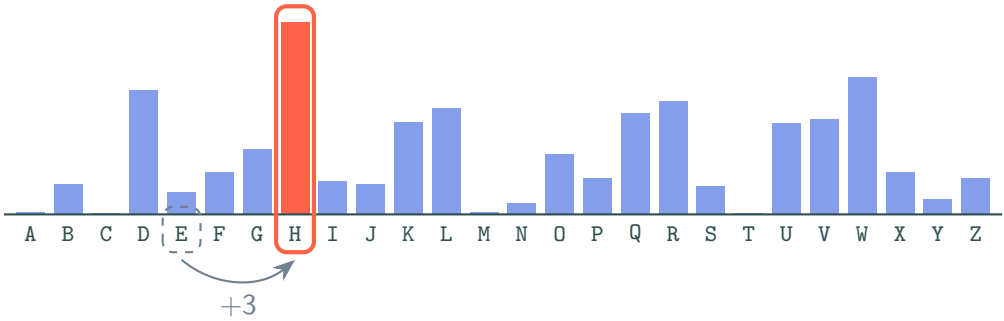
Counting Breaks Caesar Too

The tall bar is at H. H – E = 7 – 4 = **3**. The key is 3!



Counting Breaks Caesar Too

The tall bar is at H. H – E = 7 – 4 = **3**. The key is 3!



No need to try 26 keys. Count **once**, then unlock.

One Count, One Key

```
1 def crack_caesar(secret):
2     top = by_count(count_letters(secret))[0]
3     k = (ABC.index(top) - ABC.index('E')) % 26
4     return shift(secret, -k)
```

One Count, One Key

```
1 def crack_caesar(secret):
2     top = by_count(count_letters(secret))[0]
3     k = (ABC.index(top) - ABC.index('E')) % 26
4     return shift(secret, -k)
```

- **Line 2:** the most common letter of the secret.

One Count, One Key

```
1 def crack_caesar(secret):
2     top = by_count(count_letters(secret))[0]
3     k = (ABC.index(top) - ABC.index('E')) % 26
4     return shift(secret, -k)
```

- **Line 2:** the most common letter of the secret.
- **Line 3:** how far it is from E. That is the key.

One Count, One Key

```
1 def crack_caesar(secret):
2     top = by_count(count_letters(secret))[0]
3     k = (ABC.index(top) - ABC.index('E')) % 26
4     return shift(secret, -k)
```

- **Line 2:** the most common letter of the secret.
- **Line 3:** how far it is from `E`. That is the key.
- **Line 4:** move every letter back.

One Count, One Key

```
1 def crack_caesar(secret):
2     top = by_count(count_letters(secret))[0]
3     k = (ABC.index(top) - ABC.index('E')) % 26
4     return shift(secret, -k)
```

- **Line 2:** the most common letter of the secret.
- **Line 3:** how far it is from `E`. That is the key.
- **Line 4:** move every letter back.

It needs a **long** secret. In `'HI MOM'`, `E` does not even appear!

You Try! *(write your answers, 1 min)*

A long secret was locked with a shift.
Its most common letter is J.

(a) What is the key?

(b) Unlock the word MJQQT.

You Try! *(write your answers, 1 min)*

A long secret was locked with a shift.
Its most common letter is J.

(a) What is the key?

J – E = 9 – 4 = **5**

(b) Unlock the word MJQQT.

You Try! *(write your answers, 1 min)*

A long secret was locked with a shift.
Its most common letter is J.

(a) What is the key?

J – E = 9 – 4 = **5**

(b) Unlock the word MJQQT.

Each letter back 5: HELLO

Round 2



Codemaker



Codebreaker

1	Shift every letter	Try all 26 keys ✓
2	Mix the alphabet	Count the letters ✓
3	?	?
now	?	?

Round 2



Codemaker



Codebreaker

1	Shift every letter	Try all 26 keys ✓
2	Mix the alphabet	Count the letters ✓
3	?	?
now	?	?

Try 26! keys: 1.3 trillion years. Count: **one pass**, $O(n)$.

Big Idea

Clever beats **brute force**.

A lock with many keys can still be weak.

Look for a **pattern** the lock does not hide.

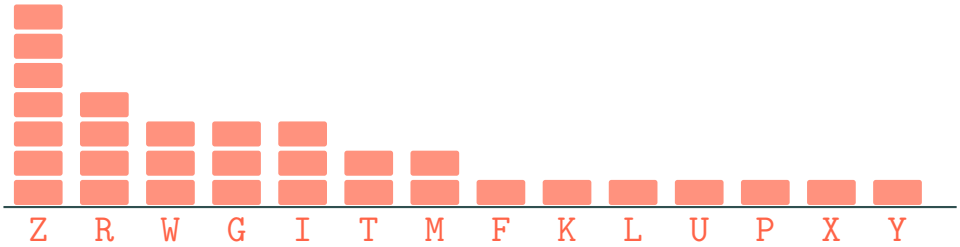
Opening the Letter

Count the Letter

WZZR WZ TR RMZ MGFIZ GK LUIPGW TR IZXZY

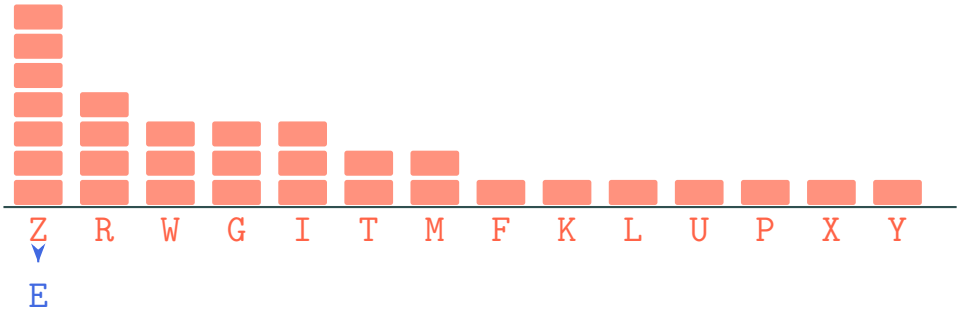
Count the Letter

WZZR WZ TR RMZ MGFIZ GK LUIPGW TR IZXZY



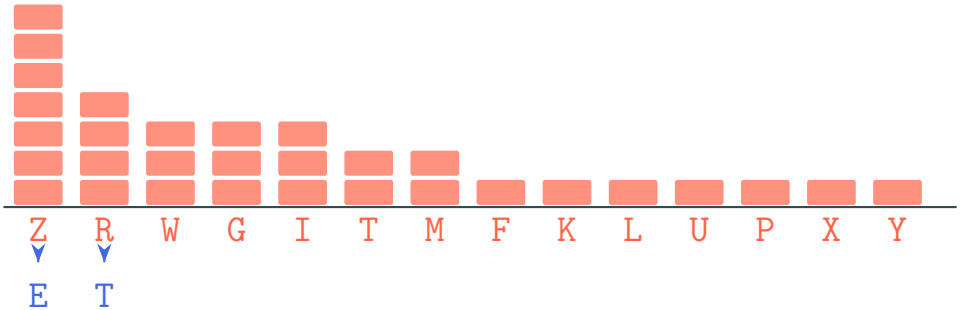
Count the Letter

WZZR WZ TR RMZ MGFIZ GK LUIPGW TR IZXZY



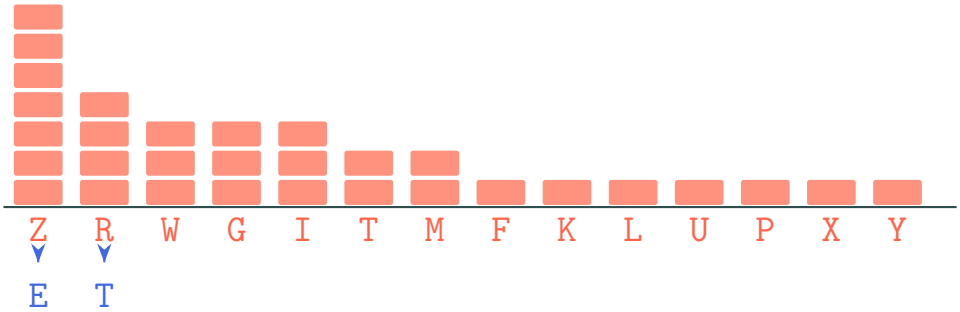
Count the Letter

WZZR WZ TR RMZ MGFIZ GK LUIPGW TR IZXZY



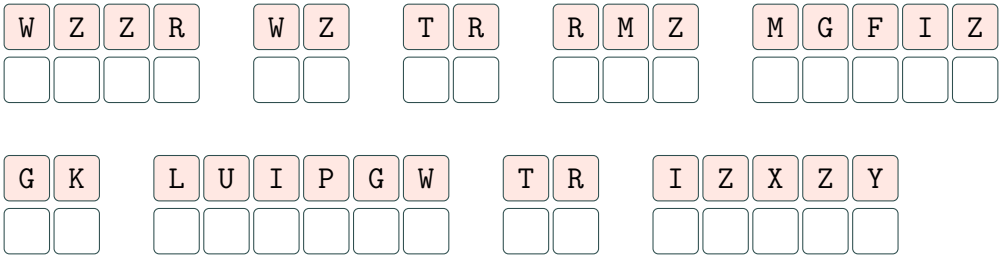
Count the Letter

WZZR WZ TR RMZ MGFIZ GK LUIPGW TR IZXZY



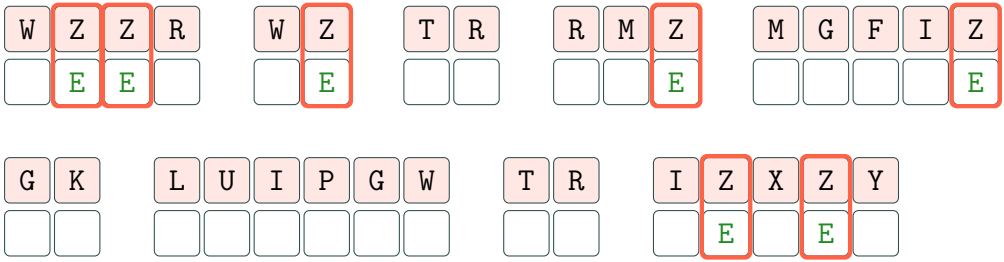
Only 38 letters, so the counts are rough. W, G, I tie. Use **words** next.

Fill In the Letters



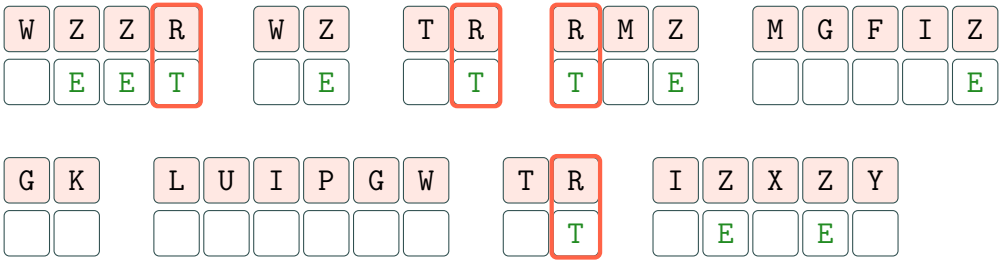
Each secret letter gets a plain letter under it.

Fill In the Letters



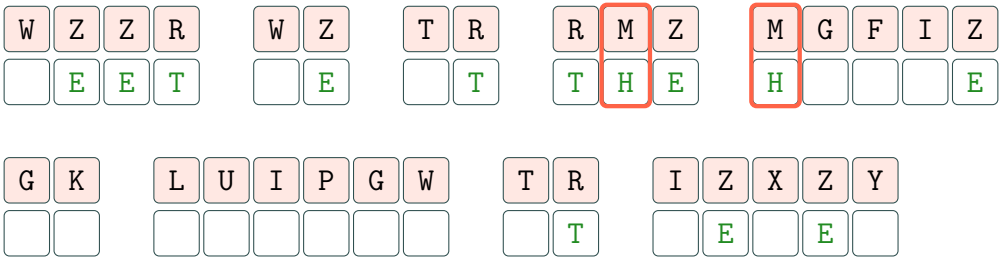
The count says: Z → E.

Fill In the Letters



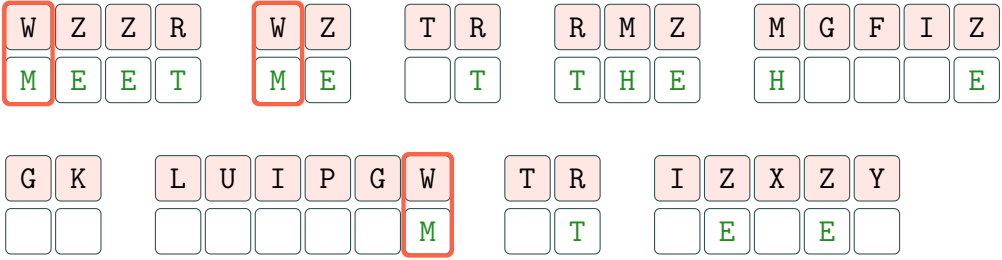
The count says: R → T.

Fill In the Letters



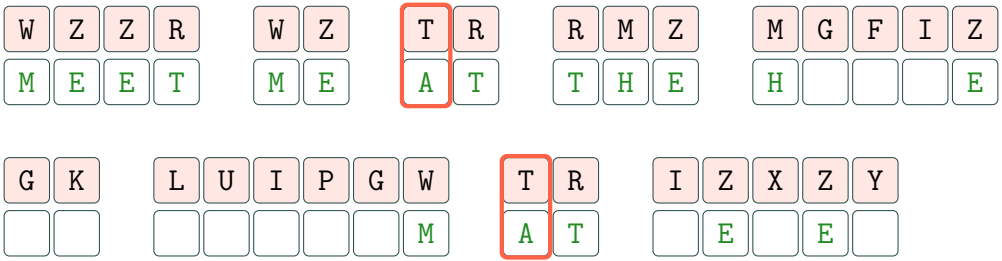
RMZ is T?E. A common word: **THE**. So M → H.

Fill In the Letters



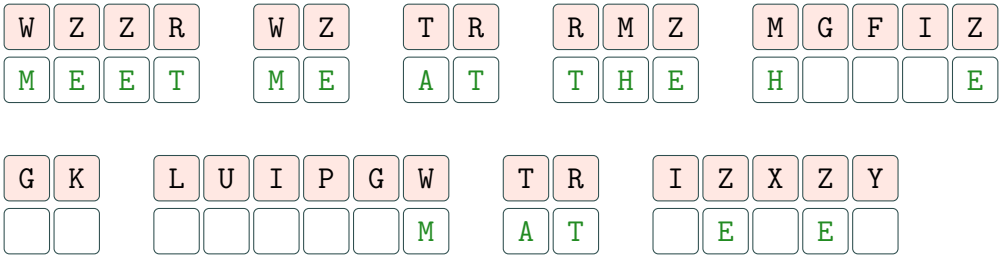
WZZR WZ is ?EET ?E: **MEET ME**. So W → M.

Fill In the Letters



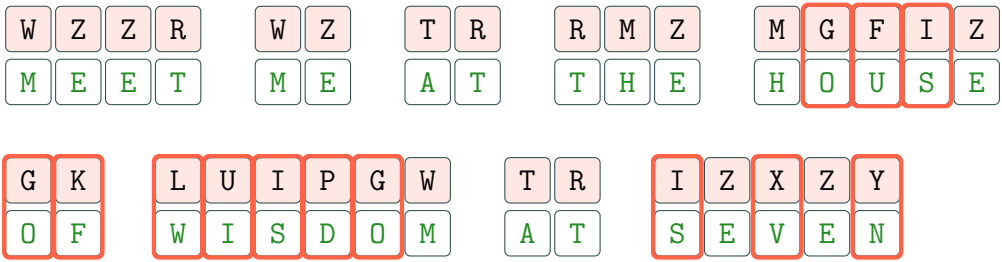
TR is ?T: **AT**. So T → A.

Fill In the Letters



You Try! *(1 min)* Now it is a word puzzle. Guess the rest!

Fill In the Letters



MEET ME AT THE HOUSE OF WISDOM AT SEVEN

Fill In the Letters

W	Z	Z	R
M	E	E	T

W	Z
M	E

T	R
A	T

R	M	Z
T	H	E

M	G	F	I	Z
H	O	U	S	E

G	K
O	F

L	U	I	P	G	W
W	I	S	D	O	M

T	R
A	T

I	Z	X	Z	Y
S	E	V	E	N

MEET ME AT THE HOUSE OF WISDOM AT SEVEN

The count gave us **2** letters. Words gave the rest. **No key** was used.

Round 3: Many Shifts

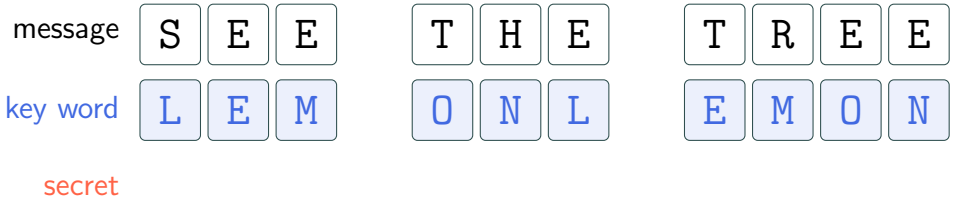
The Lock Fights Back

Italy, 1553: a **different** shift for each letter. The key is a word.

message	S	E	E		T	H	E		T	R	E	E
key word												
secret												

The Lock Fights Back

Italy, 1553: a **different** shift for each letter. The key is a word.



Key word `LEMON`: shift by `L`, then `E`, then `M`, . . . , then again.

The Lock Fights Back

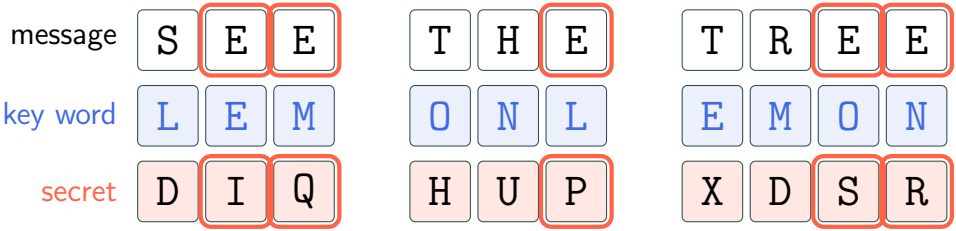
Italy, 1553: a **different** shift for each letter. The key is a word.

message	S	E	E		T	H	E		T	R	E	E
key word	L	E	M		O	N	L		E	M	O	N
secret	D	I	Q		H	U	P		X	D	S	R

Key word `LEMON`: shift by `L`, then `E`, then `M`, . . . , then again.

The Lock Fights Back

Italy, 1553: a **different** shift for each letter. The key is a word.

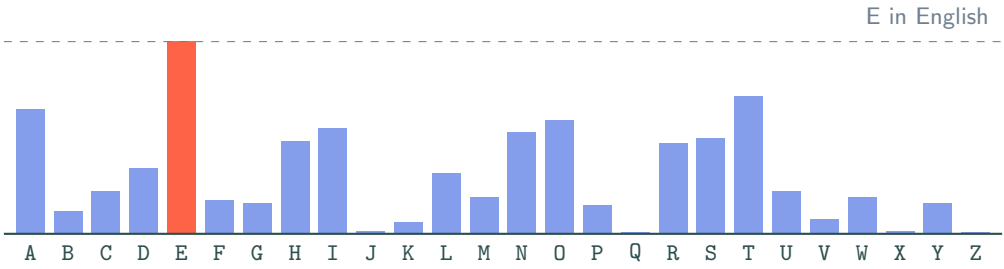


Key word LEMON: shift by L, then E, then M, . . . , then again.

The five Es became I, Q, P, S, R: **five different** letters!

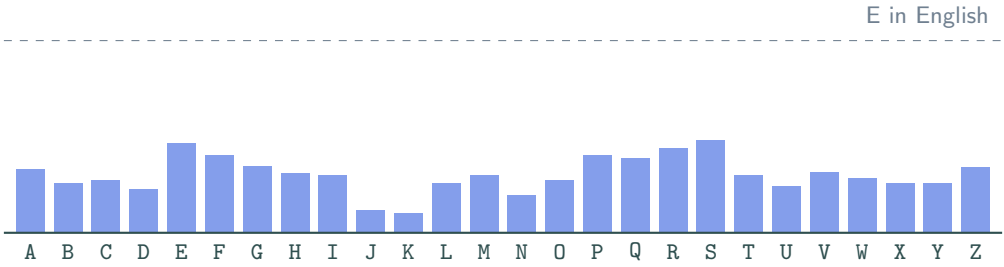
Flat Counts

English text, not locked.



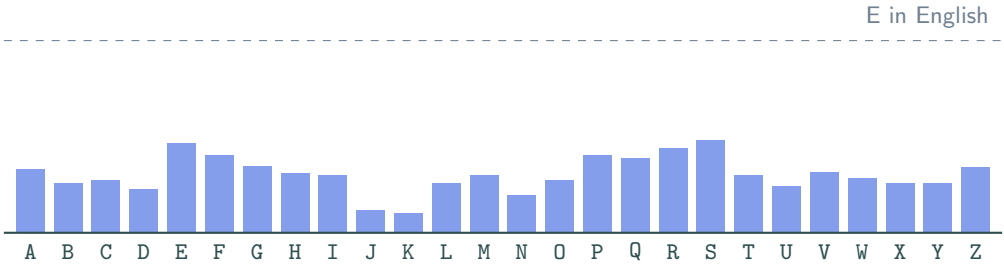
Flat Counts

The same text, locked with LEMON.



Flat Counts

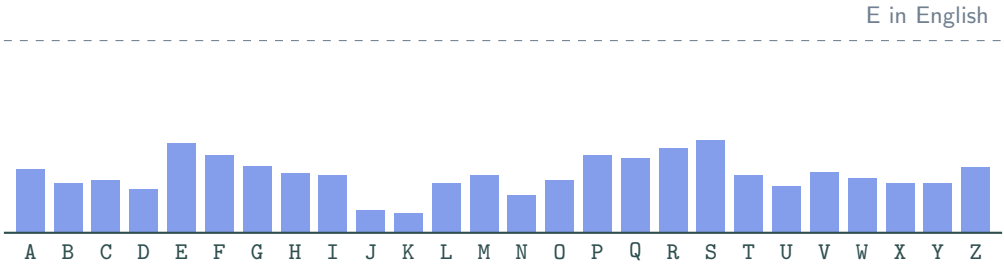
The same text, locked with `LEMON`.



- No tall bar. Al-Kindi's count finds **nothing**.

Flat Counts

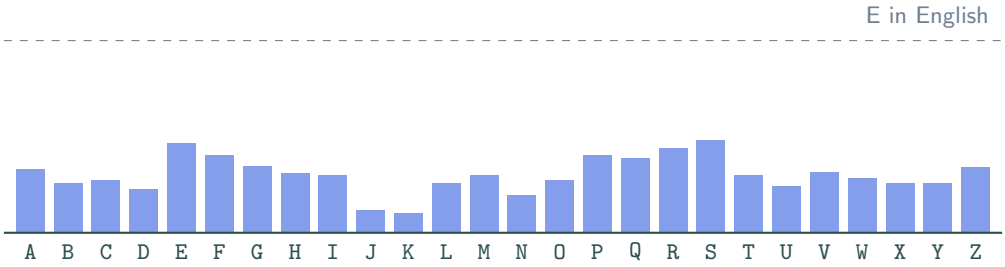
The same text, locked with LEMON.



- No tall bar. Al-Kindi's count finds **nothing**.
- Called "the code no one can break" for 300 years.

Flat Counts

The same text, locked with LEMON.



- No tall bar. Al-Kindi's count finds **nothing**.
- Called "the code no one can break" for 300 years.
- Broken in the 1800s: find the key word's **length**, then **count again**, one shift at a time.

Your Phone Today



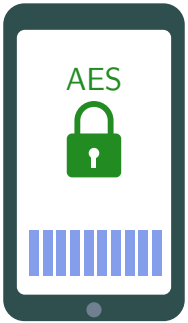
- AES (2001) locks WhatsApp and your bank.

Your Phone Today



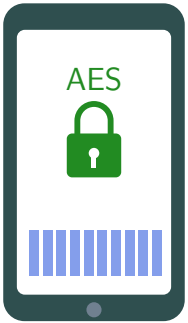
- AES (2001) locks WhatsApp and your bank.
- Keys: 2^{128} or more.

Your Phone Today



- AES (2001) locks WhatsApp and your bank.
- Keys: 2^{128} or more.
- Its secrets have **flat counts**: every value is equally common.

Your Phone Today



- AES (2001) locks WhatsApp and your bank.
- Keys: 2^{128} or more.
- Its secrets have **flat counts**: every value is equally common.

Al-Kindi's lesson, 1,200 years on:
a good lock must hide the **counts**, not just the letters.

Final Score



Codemaker



Codebreaker

1	Shift every letter	Try all 26 keys ✓
2	Mix the alphabet	Count the letters ✓
3	Many shifts	Count again ✓
now	?	?

Final Score



Codemaker



Codebreaker

	Codemaker	Codebreaker	
1	Shift every letter	Try all 26 keys	✓
2	Mix the alphabet	Count the letters	✓
3	Many shifts	Count again	✓
now ✓	AES: flat counts	?	

Summary

Summary

- **Shift:** `ABC[(i + k) % 26]`. 26 keys: try them all.
- **Mix:** `key[ABC.index(ch)]`. 26! keys: 1.3 trillion years to try.
- **Count:** a dictionary of letters, then `sorted(..., key=...)`. Most common $\leftrightarrow$ most common. One pass: $O(n)$.
- **Many shifts** flatten the counts. Modern locks keep them flat.

Big Idea

The letter had 26! keys.

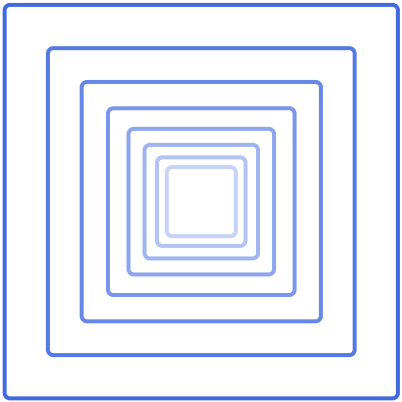
Al-Kindi tried **none** of them.

He **counted** the letters.

Clever beats brute force.

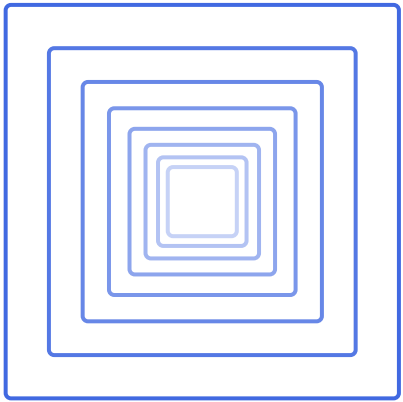
Next Time: Recursion

- Why is the website called **Stack Overflow**?



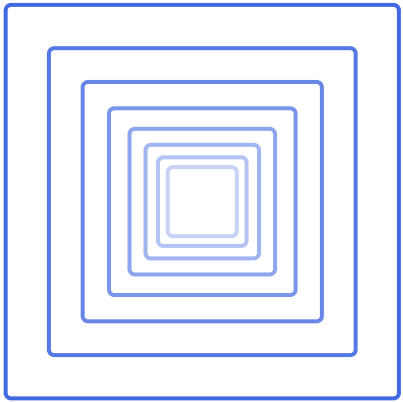
Next Time: Recursion

- Why is the website called **Stack Overflow**?
- A function that **calls itself**.



Next Time: Recursion

- Why is the website called **Stack Overflow**?
- A function that **calls itself**.
- And the day the computer runs out of room.



Questions?

Exit Ticket

Today's question *(2 minutes)*

(a) A long secret was locked with a shift.
Its most common letter is `K`.
What is the key? Unlock `ZNK`.

(b) The mixed alphabet has 26! keys.
Why did Al-Kindi not need to try them? **One** sentence.

COMP 102 — Exit Ticket #20

Name: _____

Roll #: _____ Date: _____

Answer:

Hand it to me on your way out.
No name, no attendance.