# Lecture 23: Randomness and Simulation

**Comp 102**

Fakhir Shaheen

Forman Christian University

# A Game: Roll or Stop

Try

Players take turns. First to **100** wins.

 roll: **add** it

STOP **keep** it

 **lose** the turn

 **Ali**

# A Game: Roll or Stop

Try

Players take turns. First to **100** wins.

 roll: **add** it

STOP **keep** it

 **lose** the turn

 **Ali**



4

# A Game: Roll or Stop

Try

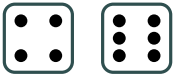
Players take turns. First to **100** wins.

 roll: **add** it

STOP **keep** it

 **lose** the turn

 **Ali**



10

# A Game: Roll or Stop

Try

Players take turns. First to **100** wins.

 roll: **add** it

STOP **keep** it

 **lose** the turn

**Ali**  13

# A Game: Roll or Stop

Try

Players take turns. First to **100** wins.

 roll: **add** it

STOP **keep** it

 **lose** the turn

score

**Ali**



STOP **keep 13**

**13**

# A Game: Roll or Stop

Try

Players take turns. First to **100** wins.

 roll: **add** it

STOP **keep** it

 **lose** the turn

score

**Ali**  STOP **keep 13** **13**

**Sana**  5

# A Game: Roll or Stop

Try

Players take turns. First to **100** wins.

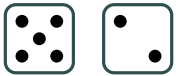
 roll: **add** it

STOP **keep** it

 **lose** the turn

score

**Ali**  STOP **keep 13** **13**

**Sana**  7

# A Game: Roll or Stop

Try

Players take turns. First to **100** wins.

 roll: **add** it

STOP **keep** it

 **lose** the turn

score

**Ali**  STOP **keep 13** 13

**Sana**  **lose 7** 0

# Play It Now! Turn 1 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**




First roll: no choice yet. Click!

# Play It Now! Turn 1 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**



6

?

This turn: **6**. **ROLL** or **STOP**?

# Play It Now! Turn 1 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**



6



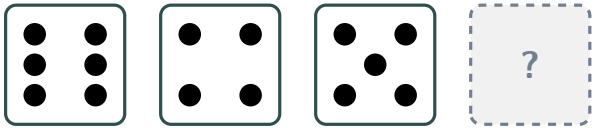
10




This turn: **10**. **ROLL** or **STOP**?

# Play It Now! Turn 1 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**




This turn: **15**. **ROLL** or **STOP**?

# Play It Now! Turn 1 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**

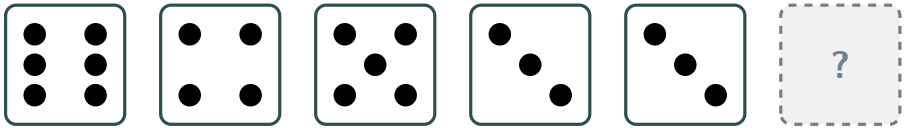



6 10 15 18

This turn: **18**. **ROLL** or **STOP**?

# Play It Now! Turn 1 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**




This turn: **21**. **ROLL** or **STOP**?

# Play It Now! Turn 1 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**




6 10 15 18 21 24

This turn: **24**. **ROLL** or **STOP**?

# Play It Now! Turn 1 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**




A **1**! Still rolling? You lose the **24**.

Stopped? **Write down** what you kept.

# Play It Now! Turn 2 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**




First roll: no choice yet. Click!

# Play It Now! Turn 2 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**



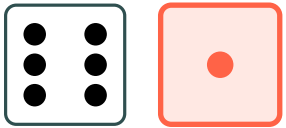
6

?

This turn: **6**. **ROLL** or **STOP**?

# Play It Now! Turn 2 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**



6 **0**

A **1**! Still rolling? You lose the **6**.

Stopped? **Write down** what you kept.

# Play It Now! Turn 3 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**




First roll: no choice yet. Click!

# Play It Now! Turn 3 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**



4

?

This turn: **4**. **ROLL** or **STOP**?

# Play It Now! Turn 3 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**

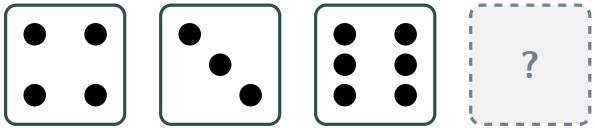



4 7

This turn: **7**. **ROLL** or **STOP**?

# Play It Now! Turn 3 of 3

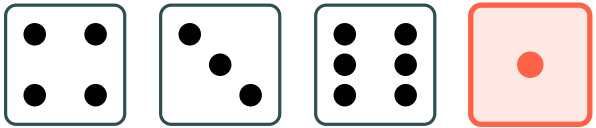
Everyone plays! Before each click, shout **ROLL** or **STOP**




This turn: **13**. **ROLL** or **STOP**?

# Play It Now! Turn 3 of 3

Everyone plays! Before each click, shout **ROLL** or **STOP**



4 7 13 **0**

A **1**! Still rolling? You lose the **13**.

Stopped? **Write down** what you kept.

# How Did You Do?

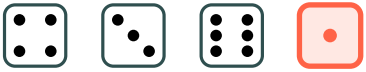
**Turn 1**




**Turn 2**




**Turn 3**



# How Did You Do?

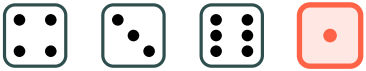
**Turn 1**




**Turn 2**




**Turn 3**




Your total? Who has the **most**?

# How Did You Do?

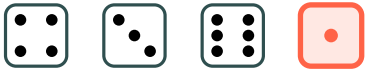
**Turn 1**




**Turn 2**




**Turn 3**



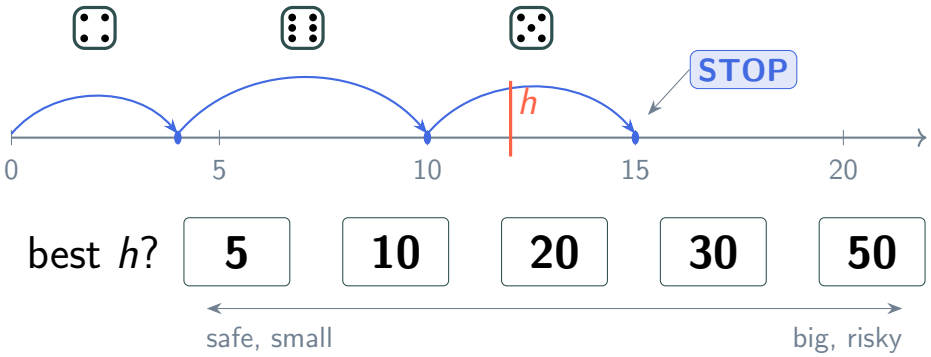

Your total? Who has the **most**?

Stopped too early? Lost it all to a 1?

# **You Try!** *(write a number, 30 s)*

one player, one turn

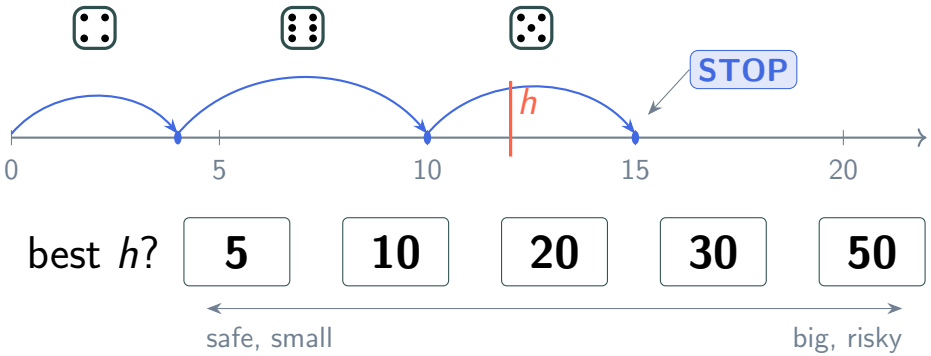
Plan: roll until this turn has **at least** $h$ points.



# You Try! *(write a number, 30 s)*

one player, one turn

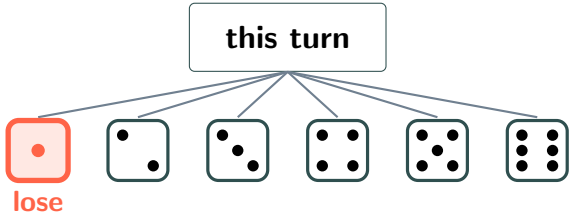
Plan: roll until this turn has **at least** $h$ points.




By the end of class we will **know**.

# Can We Work It Out?

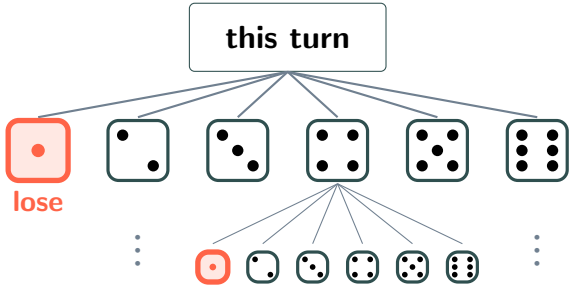
Keep rolling: every roll splits into **6**.




1 roll: 6

# Can We Work It Out?

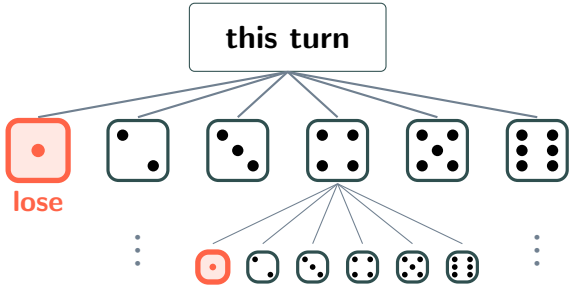
Keep rolling: every roll splits into **6**.




| | |
|---|---|
| 1 roll: | 6 |
| 2 rolls: | 36 |

# Can We Work It Out?

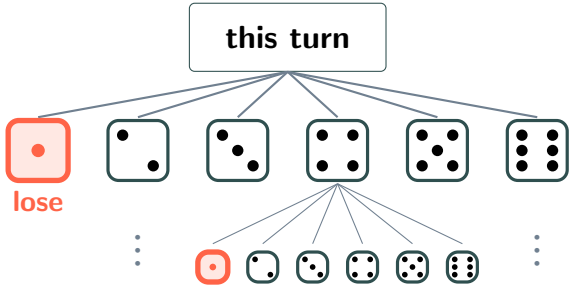
Keep rolling: every roll splits into **6**.




| | |
|---|---|
| 1 roll: | 6 |
| 2 rolls: | 36 |
| 3 rolls: | 216 |
| 10 rolls: | **60,466,176** |

# Can We Work It Out?

Keep rolling: every roll splits into **6**.

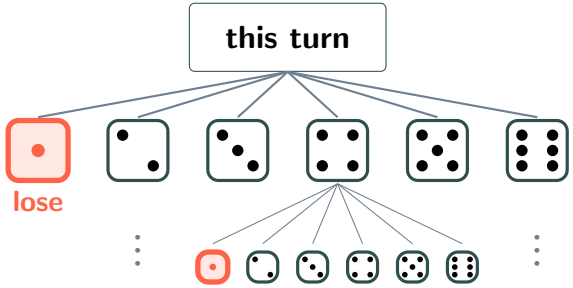



| | |
|---|---|
| 1 roll: | 6 |
| 2 rolls: | 36 |
| 3 rolls: | 216 |
| 10 rolls: | **60,466,176** |

And a game has **many** turns: far too many to list one by one!

# Can We Work It Out?

Keep rolling: every roll splits into **6**.




| | |
|---|---|
| 1 roll: | 6 |
| 2 rolls: | 36 |
| 3 rolls: | 216 |
| 10 rolls: | **60,466,176** |

And a game has **many** turns: far too many to list one by one!

Someone hit this same wall in 1946 . . .

# 1946: Ulam's Card Game

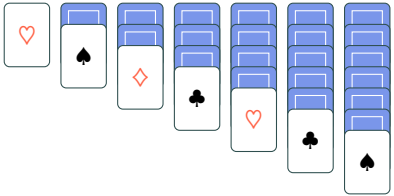


Stan **Ulam** is sick.

# 1946: Ulam's Card Game



Stan **Ulam** is sick.



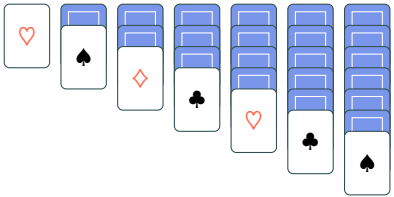
He plays **solitaire**.

# 1946: Ulam's Card Game

My chance to **win**?



Stan **Ulam** is sick.



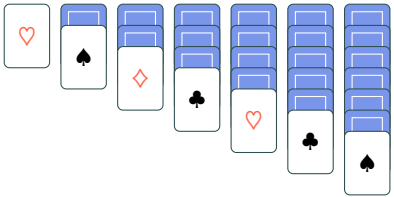
He plays **solitaire**.

# 1946: Ulam's Card Game

My chance to **win**?



Stan **Ulam** is sick.



He plays **solitaire**.

$52! \approx 8 \times 10^{67}$ ways to deal: **too many**, like our dice tree.

# Ulam's Idea

Just **play** 100 games.
Count the **wins**.

won lost

# Ulam's Idea

Just **play** 100 games.
Count the **wins**.



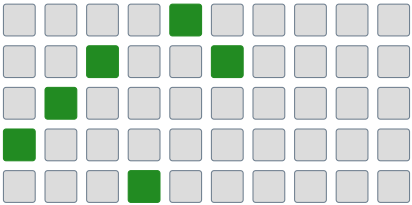
1 win in 10

won lost

# Ulam's Idea

Just **play** 100 games.
Count the **wins**.

6 wins in 50

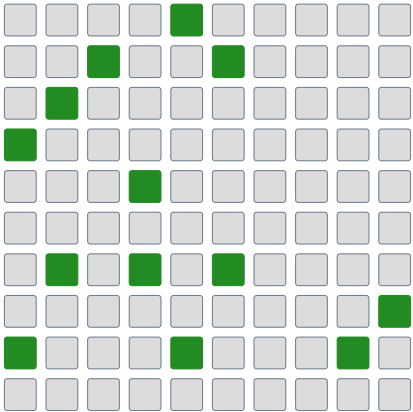


won lost

# Ulam's Idea

Just **play** 100 games.
Count the **wins**.

13 wins in 100



won lost

# Ulam's Idea

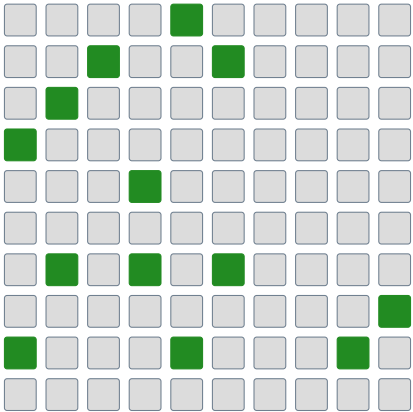
Just **play** 100 games.
Count the **wins**.

13 wins in 100

Chance ≈ **13 in 100**.

*(made-up numbers, to show the idea)*

won lost

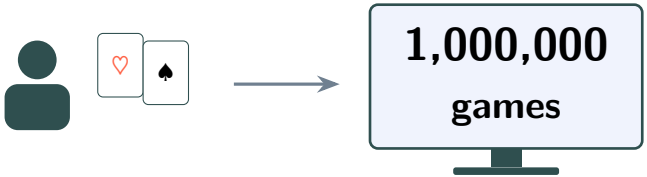


# Let a Computer Play




By hand, 100 games take **days**.

# Let a Computer Play




By hand, 100 games take **days**.

1946: the first **computers**. Let one play!

# Let a Computer Play




By hand, 100 games take **days**.

1946: the first **computers**. Let one play!

All **chance**, like the casinos of **Monte Carlo**.

# Let a Computer Play



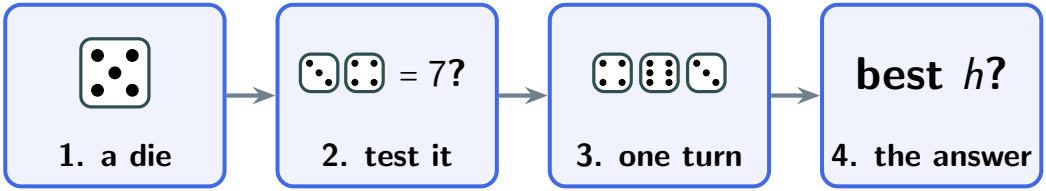

By hand, 100 games take **days**.

1946: the first **computers**. Let one play!

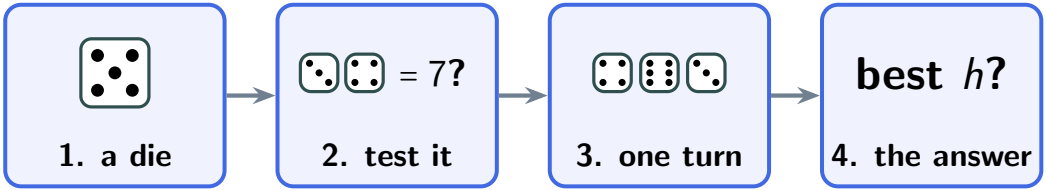
All **chance**, like the casinos of **Monte Carlo**.

**Monte Carlo method:** play many times. **Count**.

# Our Plan: Let Python Play



# Our Plan: Let Python Play




Then Python plays **thousands** of turns for each $h$.

# Roll a Die

Try

```
>>> import random
>>> random.randint(1, 6)

>>> random.randint(1, 6)

>>> random.randint(1, 6)
```



# Roll a Die

Try

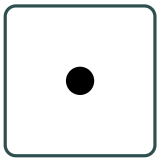
```
>>> import random
>>> random.randint(1, 6)
4
>>> random.randint(1, 6)

>>> random.randint(1, 6)
```



# Roll a Die

Try

```
>>> import random
>>> random.randint(1, 6)
4
>>> random.randint(1, 6)
1
>>> random.randint(1, 6)
```



# Roll a Die

Try

```
>>> import random
>>> random.randint(1, 6)
4
>>> random.randint(1, 6)
1
>>> random.randint(1, 6)
6
```



# Roll a Die

Try

```
>>> import random
>>> random.randint(1, 6)
4
>>> random.randint(1, 6)
1
>>> random.randint(1, 6)
6
```



Same line. A **different** answer each time.

# Roll a Die

Try

```
>>> import random
>>> random.randint(1, 6)
4
>>> random.randint(1, 6)
1
>>> random.randint(1, 6)
6
```



Same line. A **different** answer each time.

You used it in Hangman: `random.choice(wordlist)`.

# Two Tools

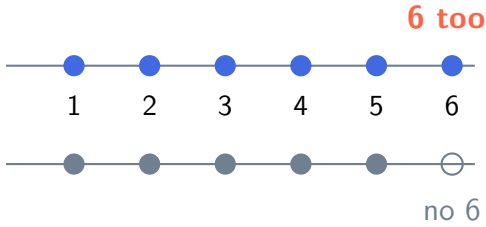
`random.randint(1, 6)`



# Two Tools

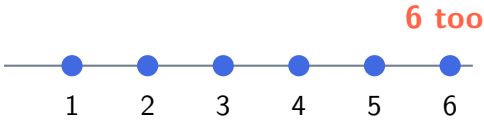
`random.randint(1, 6)`

`range(1, 6)`



# Two Tools

`random.randint(1, 6)`




`range(1, 6)`




`random.choice(['ROLL', 'STOP'])` **ROLL** or **STOP**

one item of the list

# **You Try!** *(write YES or NO for each, 1 min)*

Can this line **ever** give this value?

| line | value | ever? |
|---|---|---|
| `random.randint(1, 6)` | `6` | |
| `random.randint(1, 6)` | `0` | |
| `random.randint(1, 6)` | `3.5` | |
| `random.choice(['ROLL', 'STOP'])` | `'STOP'` | |
| `random.choice('HT')` | `'T'` | |

# **You Try!** *(write YES or NO for each, 1 min)*

Can this line **ever** give this value?

| line | value | ever? |
|---|---|---|
| `random.randint(1, 6)` | `6` | **YES** |
| `random.randint(1, 6)` | `0` | **NO** |
| `random.randint(1, 6)` | `3.5` | |
| `random.choice(['ROLL', 'STOP'])` | `'STOP'` | |
| `random.choice('HT')` | `'T'` | |

# **You Try!** *(write YES or NO for each, 1 min)*

Can this line **ever** give this value?

| line | value | ever? |
|---|---|---|
| `random.randint(1, 6)` | `6` | **YES** |
| `random.randint(1, 6)` | `0` | **NO** |
| `random.randint(1, 6)` | `3.5` | **NO** (whole numbers only) |
| `random.choice(['ROLL', 'STOP'])` | `'STOP'` | **YES** |
| `random.choice('HT')` | `'T'` | |

# **You Try!** *(write YES or NO for each, 1 min)*

Can this line **ever** give this value?

| line | value | ever? |
|---|---|---|
| `random.randint(1, 6)` | 6 | **YES** |
| `random.randint(1, 6)` | 0 | **NO** |
| `random.randint(1, 6)` | 3.5 | **NO** (whole numbers only) |
| `random.choice(['ROLL', 'STOP'])` | `'STOP'` | **YES** |
| `random.choice('HT')` | `'T'` | **YES** (a string of letters) |

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

Three friends play 8 games. Pick who goes **first** each game:

```
1  import random
2
3  names = ['Ali', 'Sana', 'Omar']
4  for i in range(8):
5      k = random.randint(0, len(names))
6      print(names[k])
```

**Symptom:**

| Run 1 | Run 2 |
|---|---|
| Sana | Ali |
| Ali | Ali |
| Omar | Ali |
| Ali | Omar |
| IndexError | Sana |
| | Omar |
| | Omar |
| | Sana |

Same code. Run 2 **works**!
How do we catch a bug that **hides**?

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

Three friends play 8 games. Pick who goes **first** each game:

```
1  import random
2
3  names = ['Ali', 'Sana', 'Omar']
4  for i in range(8):
5      k = random.randint(0, len(names))
6      print(names[k])
```

**Line 5.** `randint` keeps **both** ends, so `k` can be **3**. There is no `names[3]`.

Fix: `randint(0, len(names) - 1)`

or just `random.choice(names)`.

## Symptom:

| Run 1 | Run 2 |
|---|---|
| Sana | Ali |
| Ali | Ali |
| Omar | Ali |
| Ali | Omar |
| IndexError | Sana |
| | Omar |
| | Omar |
| | Sana |

Same code. Run 2 **works**! How do we catch a bug that **hides**?

# A Computer Cannot Throw Dice

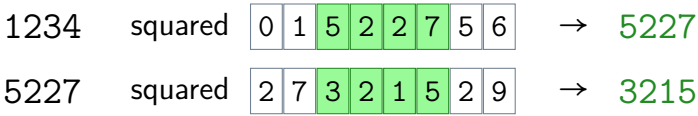
It only does **maths**. Von Neumann's trick (1949):




Square it. Keep the **middle four** digits.

# A Computer Cannot Throw Dice

It only does **maths**. Von Neumann's trick (1949):




Then do it again, with the new number.

# A Computer Cannot Throw Dice

It only does **maths**. Von Neumann's trick (1949):

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1234 | squared | 0 | 1 | 5 | 2 | 2 | 7 | 5 | 6 | → 5227 |
| 5227 | squared | 2 | 7 | 3 | 2 | 1 | 5 | 2 | 9 | → 3215 |
| 3215 | squared | 1 | 0 | 3 | 3 | 6 | 2 | 2 | 5 | → 3362 |

Then do it again, with the new number.

# A Computer Cannot Throw Dice

It only does **maths**. Von Neumann's trick (1949):

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1234 | squared | 0 | 1 | 5 | 2 | 2 | 7 | 5 | 6 | → 5227 |
| 5227 | squared | 2 | 7 | 3 | 2 | 1 | 5 | 2 | 9 | → 3215 |
| 3215 | squared | 1 | 0 | 3 | 3 | 6 | 2 | 2 | 5 | → 3362 |
| 3362 | squared | 1 | 1 | 3 | 0 | 3 | 0 | 4 | 4 | → 3030 |
| 3030 | squared | 0 | 9 | 1 | 8 | 0 | 9 | 0 | 0 | → 1809 |

5227, 3215, 3362, 3030, 1809... they **look** random!

# A Computer Cannot Throw Dice

It only does **maths**. Von Neumann's trick (1949):

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1234 | squared | 0 | 1 | 5 | 2 | 2 | 7 | 5 | 6 | → 5227 |
| 5227 | squared | 2 | 7 | 3 | 2 | 1 | 5 | 2 | 9 | → 3215 |
| 3215 | squared | 1 | 0 | 3 | 3 | 6 | 2 | 2 | 5 | → 3362 |
| 3362 | squared | 1 | 1 | 3 | 0 | 3 | 0 | 4 | 4 | → 3030 |
| 3030 | squared | 0 | 9 | 1 | 8 | 0 | 9 | 0 | 0 | → 1809 |

Run it again from 1234: the **same** numbers come out.

# A Computer Cannot Throw Dice

It only does **maths**. Von Neumann's trick (1949):

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1234 | squared | 0 | 1 | 5 | 2 | 2 | 7 | 5 | 6 | → 5227 |
| 5227 | squared | 2 | 7 | 3 | 2 | 1 | 5 | 2 | 9 | → 3215 |
| 3215 | squared | 1 | 0 | 3 | 3 | 6 | 2 | 2 | 5 | → 3362 |
| 3362 | squared | 1 | 1 | 3 | 0 | 3 | 0 | 4 | 4 | → 3030 |
| 3030 | squared | 0 | 9 | 1 | 8 | 0 | 9 | 0 | 0 | → 1809 |

Run it again from 1234: the **same** numbers come out.

They only **look** random: **pseudo-random**.

# The Same “Random” Every Time

```
random.seed(7)
for i in range(8):
    print(random.randint(1, 6), end=' ')
```

**Run 1**

# The Same "Random" Every Time

```
random.seed(7)
for i in range(8):
    print(random.randint(1, 6), end=' ')
```

**Run 1**

```
3 2 4 6 1 1 5 1
```

# The Same "Random" Every Time

```
random.seed(7)
for i in range(8):
    print(random.randint(1, 6), end=' ')
```

**Run 1**

```
3 2 4 6 1 1 5 1
```

**Run 2 (later)**

```
3 2 4 6 1 1 5 1
```

# The Same "Random" Every Time

```
random.seed(7)
for i in range(8):
    print(random.randint(1, 6), end=' ')
```

**Run 1** `3 2 4 6 1 1 5 1`

**Run 2 (later)** `3 2 4 6 1 1 5 1`

The **seed** is the start number, like 1234.

# The Same "Random" Every Time

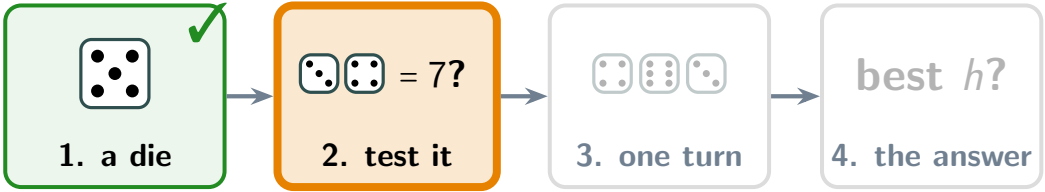
```
random.seed(7)
for i in range(8):
    print(random.randint(1, 6), end=' ')
```

**Run 1** `3 2 4 6 1 1 5 1`

**Run 2 (later)** `3 2 4 6 1 1 5 1`

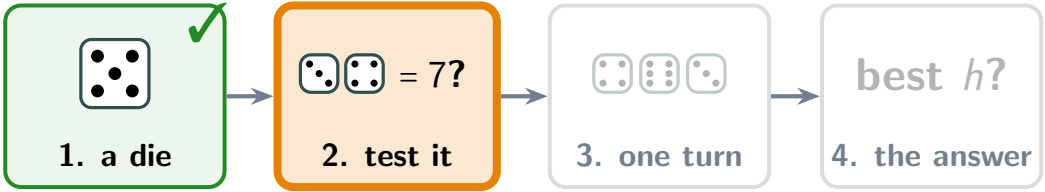
The **seed** is the start number, like 1234.
Bug Hunt + `random.seed(1)`: the crash comes **every** run.

# Step 2 of 4




Python can roll a die. Now try Ulam's recipe.

# Step 2 of 4




Python can roll a die. Now try Ulam's recipe.
First on a question we **can** answer by hand.

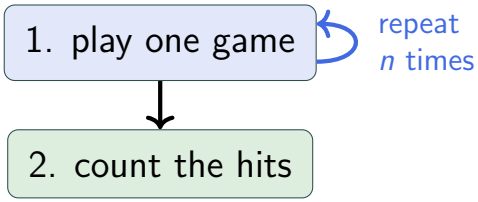
# Ulam's Recipe, in Python

```
hits = 0
for i in range(n):
    play one game
    if it worked:
        hits += 1
print(hits / n)
```

1. play one game

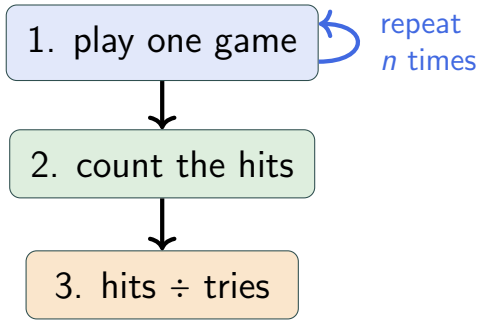
# Ulam's Recipe, in Python

```
hits = 0
for i in range(n):
    play one game
    if it worked:
        hits += 1
print(hits / n)
```



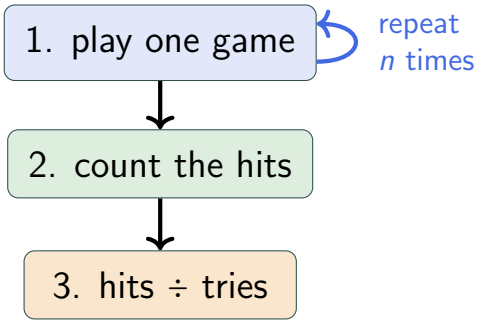
# Ulam's Recipe, in Python

```
hits = 0
for i in range(n):
    play one game
    if it worked:
        hits += 1
print(hits / n)
```



# Ulam's Recipe, in Python

```
hits = 0
for i in range(n):
    play one game
    if it worked:
        hits += 1
print(hits / n)
```




This is a **simulation**: a game played inside the computer.

# Two Dice: Total of 7?

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |

**By hand:**

36 pairs.

# Two Dice: Total of 7?

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | **7** |
| 2 | 3 | 4 | 5 | 6 | **7** | 8 |
| 3 | 4 | 5 | 6 | **7** | 8 | 9 |
| 4 | 5 | 6 | **7** | 8 | 9 | 10 |
| 5 | 6 | **7** | 8 | 9 | 10 | 11 |
| 6 | **7** | 8 | 9 | 10 | 11 | 12 |

**By hand:**

36 pairs.

**6** of them make 7.

# Two Dice: Total of 7?

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | **7** |
| 2 | 3 | 4 | 5 | 6 | **7** | 8 |
| 3 | 4 | 5 | 6 | **7** | 8 | 9 |
| 4 | 5 | 6 | **7** | 8 | 9 | 10 |
| 5 | 6 | **7** | 8 | 9 | 10 | 11 |
| 6 | **7** | 8 | 9 | 10 | 11 | 12 |

**By hand:**

36 pairs.

**6** of them make 7.

$$\frac{6}{36} = 0.1667$$

# Two Dice: Total of 7?

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | **7** |
| 2 | 3 | 4 | 5 | 6 | **7** | 8 |
| 3 | 4 | 5 | 6 | **7** | 8 | 9 |
| 4 | 5 | 6 | **7** | 8 | 9 | 10 |
| 5 | 6 | **7** | 8 | 9 | 10 | 11 |
| 6 | **7** | 8 | 9 | 10 | 11 | 12 |

**By hand:**

36 pairs.

**6** of them make 7.

$$\frac{6}{36} = 0.1667$$

Does the recipe find it?

# Two Dice, Simulated

Try

```
def chance_of_seven(n):
    hits = 0
    for i in range(n):
        a = random.randint(1, 6)
        b = random.randint(1, 6)
        if a + b == 7:
            hits += 1
    return hits / n
```

| tries | answer |
|---|---|
| 10 | |
| 100 | |
| 1,000 | |
| 10,000 | |
| 100,000 | |
| 1,000,000 | |
| exact | 0.1667 |

# Two Dice, Simulated

Try

```
def chance_of_seven(n):
    hits = 0
    for i in range(n):
        a = random.randint(1, 6)
        b = random.randint(1, 6)
        if a + b == 7:
            hits += 1
    return hits / n
```

| tries | answer |
|---|---|
| 10 | 0.2 |
| 100 | |
| 1,000 | |
| 10,000 | |
| 100,000 | |
| 1,000,000 | |
| exact | 0.1667 |

# Two Dice, Simulated

Try

```
def chance_of_seven(n):
    hits = 0
    for i in range(n):
        a = random.randint(1, 6)
        b = random.randint(1, 6)
        if a + b == 7:
            hits += 1
    return hits / n
```

| tries | answer |
|---|---|
| 10 | 0.2 |
| 100 | 0.2 |
| 1,000 | |
| 10,000 | |
| 100,000 | |
| 1,000,000 | |
| exact | 0.1667 |

# Two Dice, Simulated

Try

```
def chance_of_seven(n):
    hits = 0
    for i in range(n):
        a = random.randint(1, 6)
        b = random.randint(1, 6)
        if a + b == 7:
            hits += 1
    return hits / n
```

| tries | answer |
|---|---|
| 10 | 0.2 |
| 100 | 0.2 |
| 1,000 | 0.17 |
| 10,000 | |
| 100,000 | |
| 1,000,000 | |
| exact | 0.1667 |

# Two Dice, Simulated

Try

```
def chance_of_seven(n):
    hits = 0
    for i in range(n):
        a = random.randint(1, 6)
        b = random.randint(1, 6)
        if a + b == 7:
            hits += 1
    return hits / n
```

| tries | answer |
|---|---|
| 10 | 0.2 |
| 100 | 0.2 |
| 1,000 | 0.17 |
| 10,000 | 0.1588 |
| 100,000 | |
| 1,000,000 | |
| exact | 0.1667 |

# Two Dice, Simulated

Try

```
def chance_of_seven(n):
    hits = 0
    for i in range(n):
        a = random.randint(1, 6)
        b = random.randint(1, 6)
        if a + b == 7:
            hits += 1
    return hits / n
```

| tries | answer |
|---|---|
| 10 | 0.2 |
| 100 | 0.2 |
| 1,000 | 0.17 |
| 10,000 | 0.1588 |
| 100,000 | 0.16608 |
| 1,000,000 | |
| exact | 0.1667 |

# Two Dice, Simulated

Try

```
def chance_of_seven(n):
    hits = 0
    for i in range(n):
        a = random.randint(1, 6)
        b = random.randint(1, 6)
        if a + b == 7:
            hits += 1
    return hits / n
```

| tries | answer |
|---|---|
| 10 | 0.2 |
| 100 | 0.2 |
| 1,000 | 0.17 |
| 10,000 | 0.1588 |
| 100,000 | 0.16608 |
| 1,000,000 | 0.167684 |
| exact | 0.1667 |

# Two Dice, Simulated

Try

```
def chance_of_seven(n):
    hits = 0
    for i in range(n):
        a = random.randint(1, 6)
        b = random.randint(1, 6)
        if a + b == 7:
            hits += 1
    return hits / n
```

| tries | answer |
|---|---|
| 10 | 0.2 |
| 100 | 0.2 |
| 1,000 | 0.17 |
| 10,000 | 0.1588 |
| 100,000 | 0.16608 |
| 1,000,000 | 0.167684 |
| exact | 0.1667 |

More tries → **closer**. The recipe works!

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

A friend's version:

```
1  def chance_of_seven(n):
2      hits = 0
3      a = random.randint(1, 6)
4      b = random.randint(1, 6)
5      for i in range(n):
6          if a + b == 7:
7              hits += 1
8      return hits / n
9
10 print(chance_of_seven(1000))
```

**Symptom:** four runs:

```
0.0    0.0
1.0    0.0
```

Never near 0.1667. Only **0** or **1**!

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

A friend's version:

```
1  def chance_of_seven(n):
2      hits = 0
3      a = random.randint(1, 6)
4      b = random.randint(1, 6)
5      for i in range(n):
6          if a + b == 7:
7              hits += 1
8      return hits / n
9
10 print(chance_of_seven(1000))
```

**Symptom:** four runs:

```
0.0    0.0
1.0    0.0
```

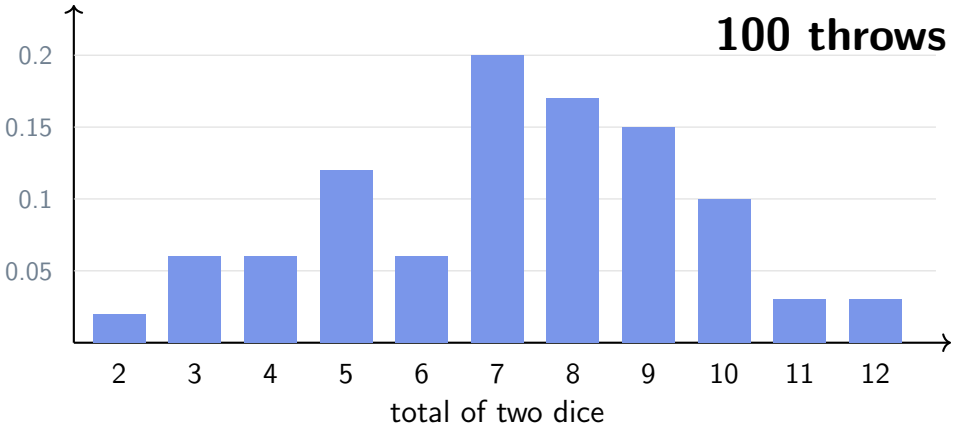
Never near 0.1667. Only **0** or **1**!

**Lines 3, 4.** The dice are rolled **once**, before the loop. All 1,000 tries use the **same** two dice.

Fix: move lines 3 and 4 **inside** the loop.

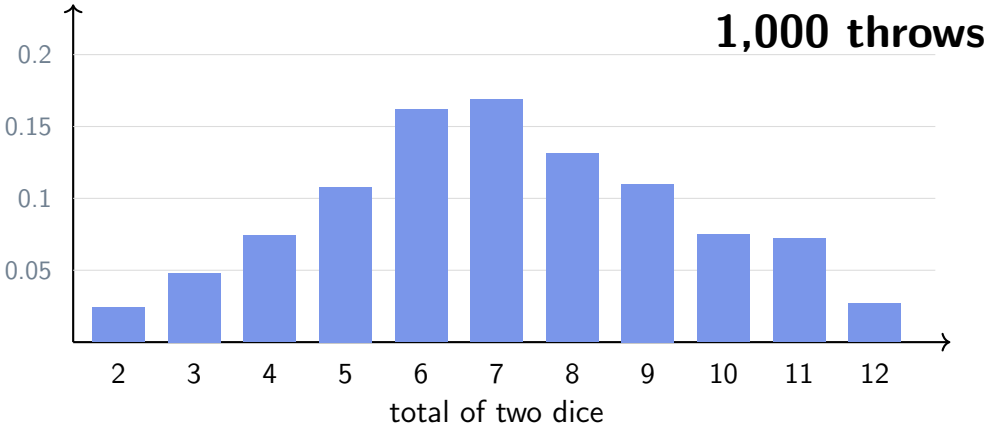
# Watch the Shape Appear

Try



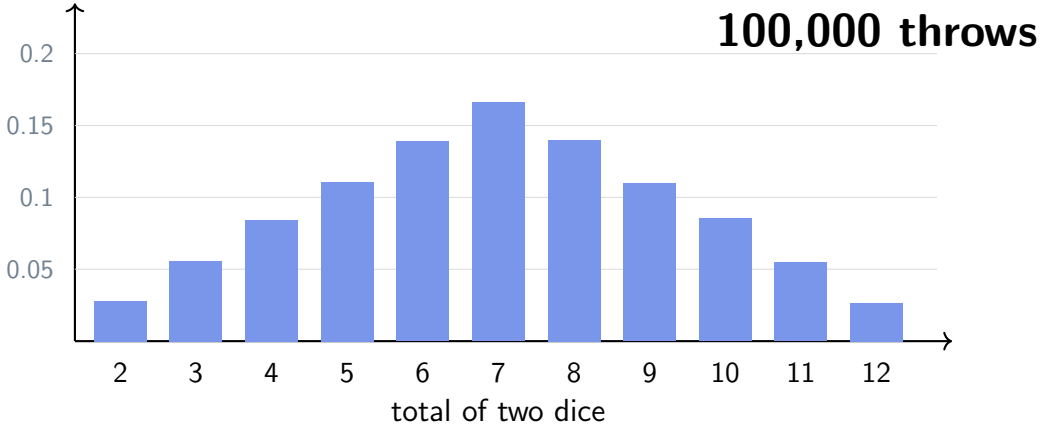
# Watch the Shape Appear

Try



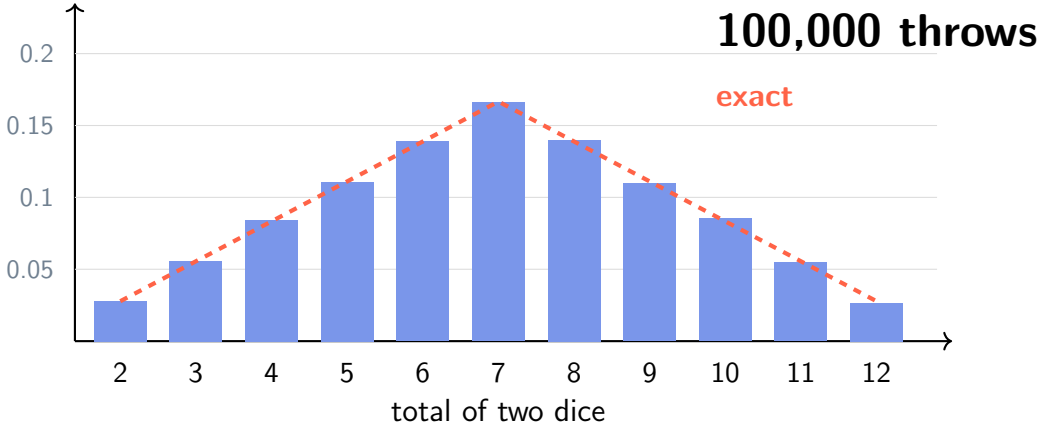
# Watch the Shape Appear

Try



# Watch the Shape Appear

Try



# How Many Tries?

Try

100 tries $\rightarrow$ 0.2

# How Many Tries?

Try

$$100 \text{ tries} \rightarrow 0.2$$

$$10{,}000 \text{ tries} \rightarrow 0.16$$

# How Many Tries?

Try

100 tries → 0.2

10,000 tries → 0.16

1,000,000 tries → 0.167

# How Many Tries?

Try

100 tries → 0.2

10,000 tries → 0.16

1,000,000 tries → 0.167

**100** × more tries → about **1 more** correct digit

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

**(a)** Sara runs `chance_of_seven(10)` and gets `0.4`.
Is her code wrong?

**(b)** 1,000,000 tries gave 3 correct digits.
About how many tries for **4** correct digits?

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

**(a)** Sara runs `chance_of_seven(10)` and gets `0.4`.
Is her code wrong?

**Not for sure.** 10 tries are far too few. 4 sevens in 10 can happen.

**(b)** 1,000,000 tries gave 3 correct digits.
About how many tries for **4** correct digits?

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

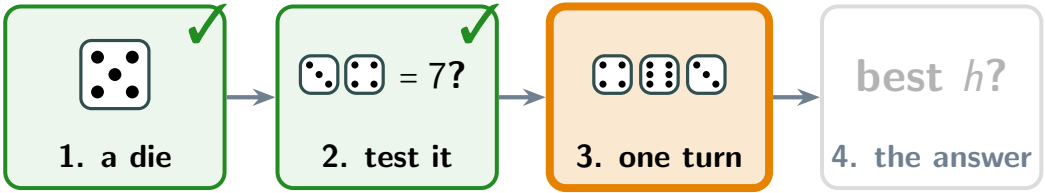
**(a)** Sara runs `chance_of_seven(10)` and gets `0.4`.
Is her code wrong?

**Not for sure.** 10 tries are far too few. 4 sevens in 10 can happen.

**(b)** 1,000,000 tries gave 3 correct digits.
About how many tries for **4** correct digits?

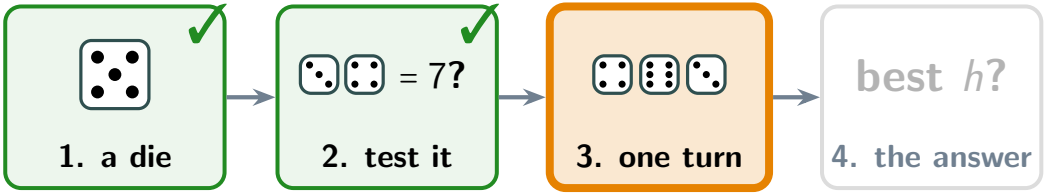
**100,000,000** (100 times more).

# Step 3 of 4




The recipe works: 0.1667, where we knew the answer.

# Step 3 of 4




The recipe works: 0.1667, where we knew the answer.

Back to Roll or Stop: teach Python **one whole turn**.

# One Turn in Python

```
1  def turn(h):
2      total = 0
3      while total < h:
4          roll = random.randint(1, 6)
5          if roll == 1:
6              return 0
7          total += roll
8      return total
```

# One Turn in Python

```
1  def turn(h):
2      total = 0
3      while total < h:
4          roll = random.randint(1, 6)
5          if roll == 1:
6              return 0
7          total += roll
8      return total
```

**Line 3:** roll until this turn has $h$ points.

# One Turn in Python

```
1  def turn(h):
2      total = 0
3      while total < h:
4          roll = random.randint(1, 6)
5          if roll == 1:
6              return 0
7          total += roll
8      return total
```



**Line 3:** roll until this turn has $h$ points.

**Lines 5–6:** a 1? This turn gives **0**.

# One Turn in Python

```
1  def turn(h):
2      total = 0
3      while total < h:
4          roll = random.randint(1, 6)
5          if roll == 1:
6              return 0
7          total += roll
8      return total
```

**Line 3:** roll until this turn has $h$ points.

**Lines 5–6:** a 1? This turn gives **0**.

**Line 7:** add the roll.

# One Turn in Python

```
1  def turn(h):
2      total = 0
3      while total < h:
4          roll = random.randint(1, 6)
5          if roll == 1:
6              return 0
7          total += roll
8      return total
```

**Line 3:** roll until this turn has $h$ points.

**Lines 5–6:** a 1? This turn gives **0**.

**Line 7:** add the roll.

**Line 8:** stop, and keep the points.

# One Turn, Three Times

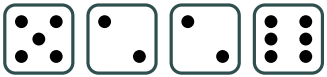
Plan: $h = 10$. What happened inside each call:

```
>>> turn(10)
```

# One Turn, Three Times

Plan: $h = 10$. What happened inside each call:

```
>>> turn(10)
```

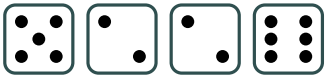


15 at least 10: stop

# One Turn, Three Times

Plan: $h = 10$. What happened inside each call:

```
>>> turn(10)
```



15 at least 10: stop

```
>>> turn(10)
```

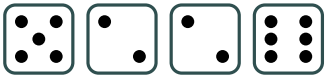


0 lost the 8

# One Turn, Three Times

Plan: $h = 10$. What happened inside each call:

```
>>> turn(10)
```



15 at least 10: stop

```
>>> turn(10)
```



0 lost the 8

```
>>> turn(10)
```

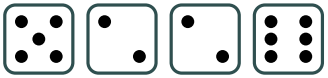


10 exactly 10: stop

# One Turn, Three Times

Plan: $h = 10$. What happened inside each call:

```
>>> turn(10)
```



15 at least 10: stop

```
>>> turn(10)
```



0 lost the 8

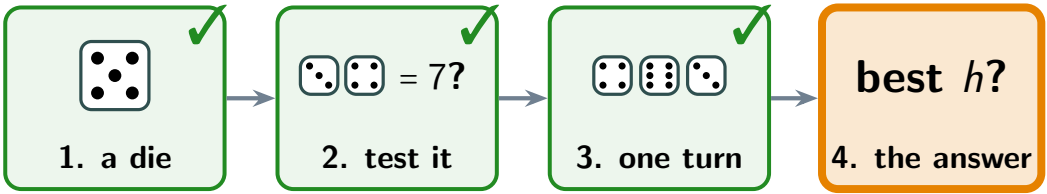
```
>>> turn(10)
```



10 exactly 10: stop

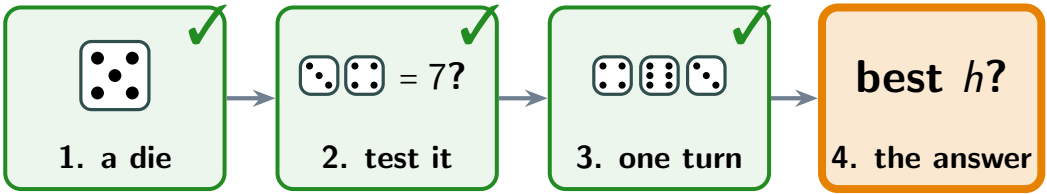
Same $h$, three scores: luck. Use the recipe on **thousands** of turns.

# Step 4 of 4




Python can play one turn.

# Step 4 of 4




Python can play one turn.

Now the recipe, where **nobody** knows the answer: the best $h$.

# Many Turns in Python

Try

```
def average_points(h, n):
    total = 0
    for i in range(n):
        total += turn(h)
    return total / n
```

play once

count

divide

# Many Turns in Python

Try

```
def average_points(h, n):
    total = 0
    for i in range(n):
        total += turn(h)
    return total / n
```

play once

count

divide

`turn(h)`

# Many Turns in Python

Try

```
def average_points(h, n):
    total = 0
    for i in range(n):
        total += turn(h)
    return total / n
```

play once — `turn(h)`

count — add the points

divide

# Many Turns in Python

Try

```
def average_points(h, n):
    total = 0
    for i in range(n):
        total += turn(h)
    return total / n
```

play once — `turn(h)`

count — add the points

divide — `total / n`

# Many Turns in Python

Try

```
def average_points(h, n):
    total = 0
    for i in range(n):
        total += turn(h)
    return total / n
```

| | |
|---|---|
| play once | `turn(h)` |
| count | add the points |
| divide | `total / n` |

The **same** recipe: add up **points**, not hits.

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

Try

A friend wrote `turn` like this:

```
1  def turn(h):
2      total = 0
3      while total < h:
4          roll = random.randint(1, 6)
5          if roll == 1:
6              break
7          total += roll
8      return total
```

**Symptom:** 100,000 turns each:

| $h$ | points per turn |
|---|---|
| 10 | 8.2 |
| 20 | 12.5 |
| 40 | 17.0 |
| 100 | 19.7 |

Bigger is **always** better? Never stop?! Too good to be true.

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

Try

A friend wrote `turn` like this:

```
1  def turn(h):
2      total = 0
3      while total < h:
4          roll = random.randint(1, 6)
5          if roll == 1:
6              break
7          total += roll
8      return total
```

**Line 6.** `break` stops the loop, but line 8 still returns the points. A 1 must lose them:

```
if roll == 1:
    return 0
```

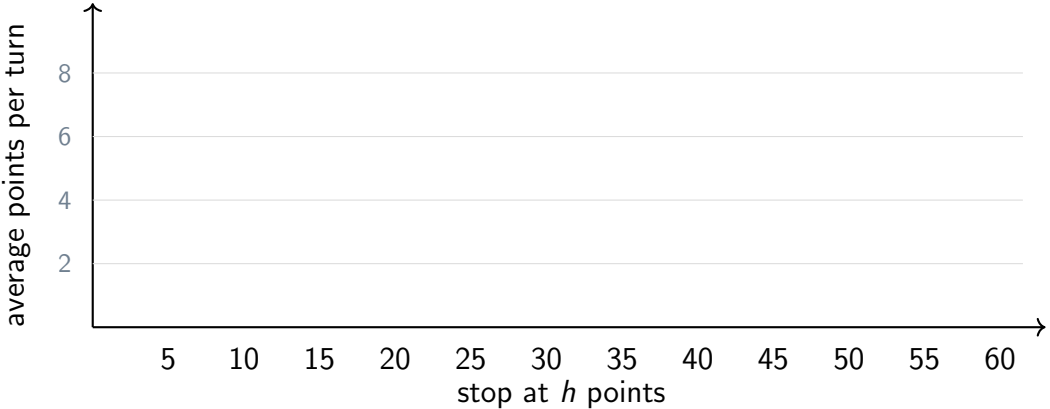
**Symptom:** 100,000 turns each:

| $h$ | points per turn |
|---|---|
| 10 | 8.2 |
| 20 | 12.5 |
| 40 | 17.0 |
| 100 | 19.7 |

Bigger is **always** better? Never stop?! Too good to be true.

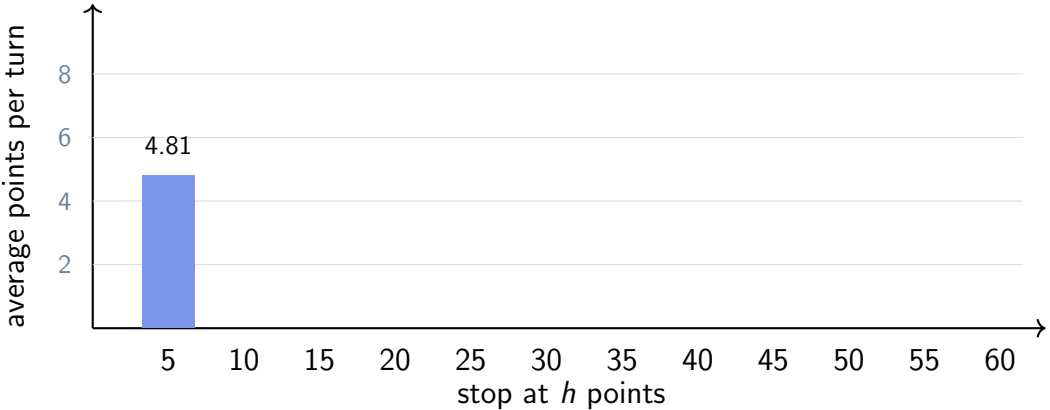
# Fixed: Points per Turn

Try



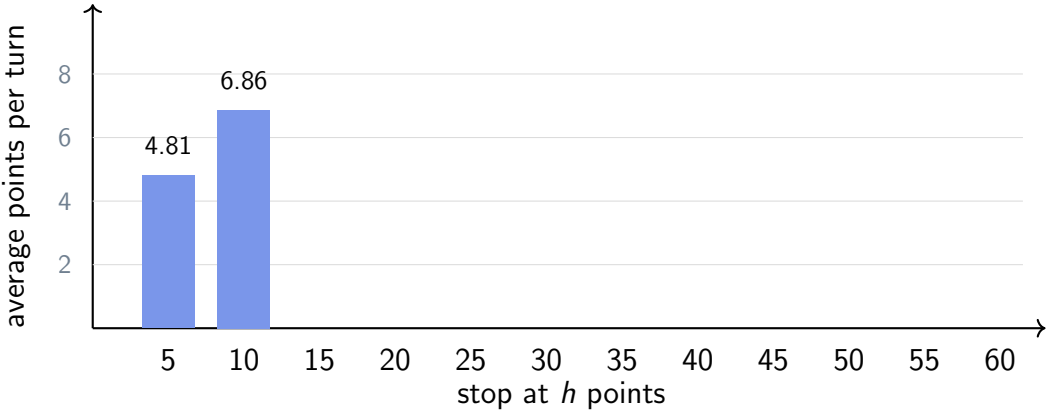
# Fixed: Points per Turn

Try



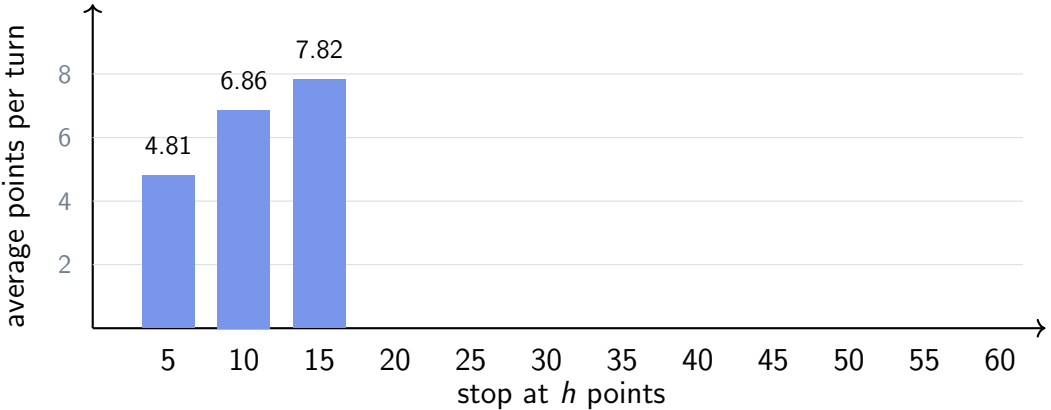
# Fixed: Points per Turn

Try



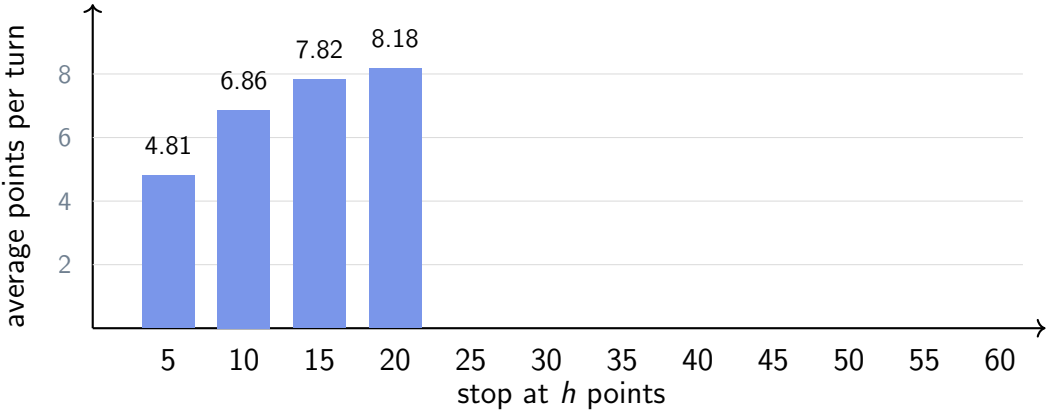
# Fixed: Points per Turn

Try



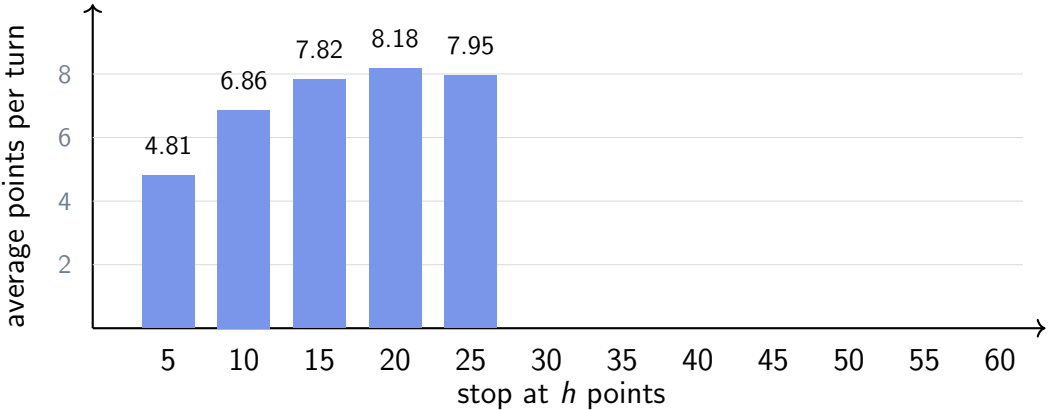
# Fixed: Points per Turn

Try



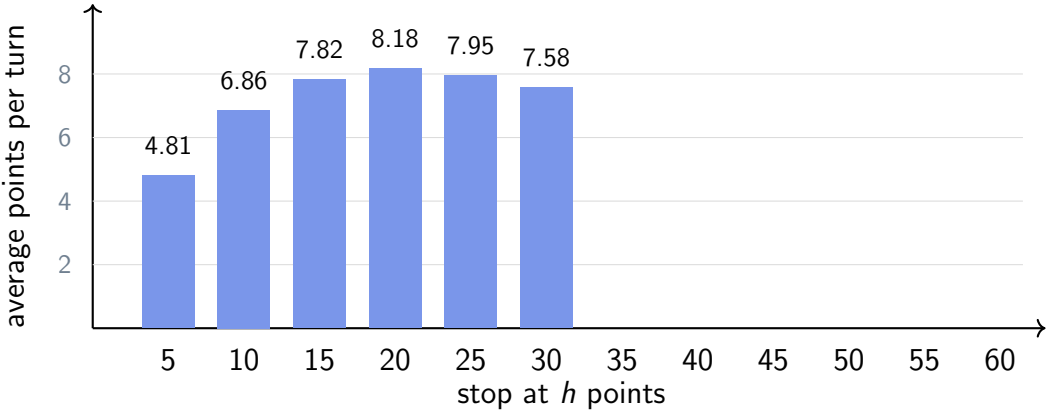
# Fixed: Points per Turn

Try



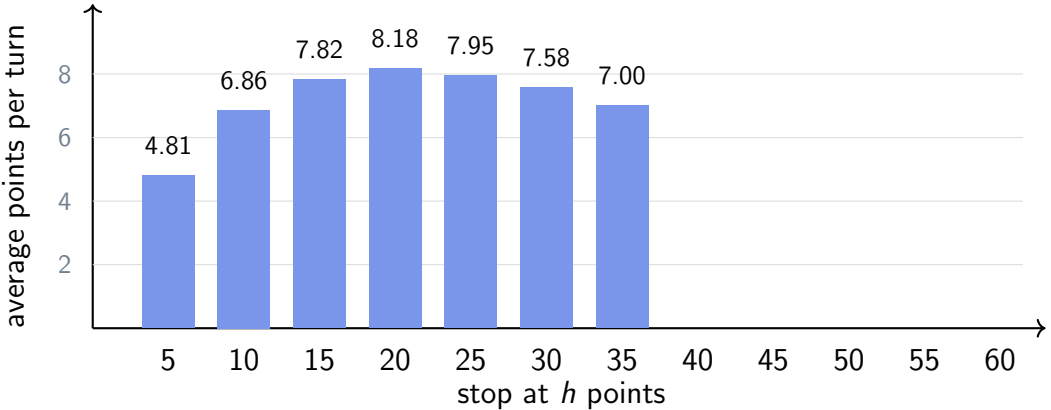
# Fixed: Points per Turn

Try



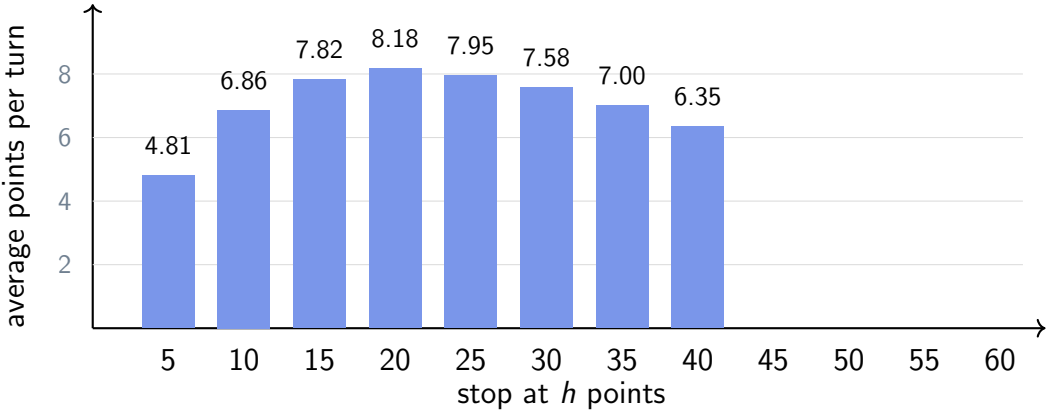
# Fixed: Points per Turn

Try



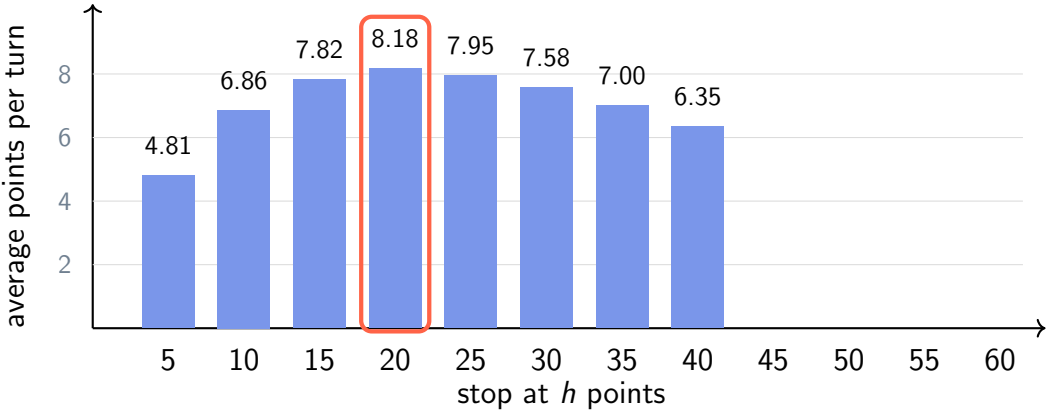
# Fixed: Points per Turn

Try



# Fixed: Points per Turn

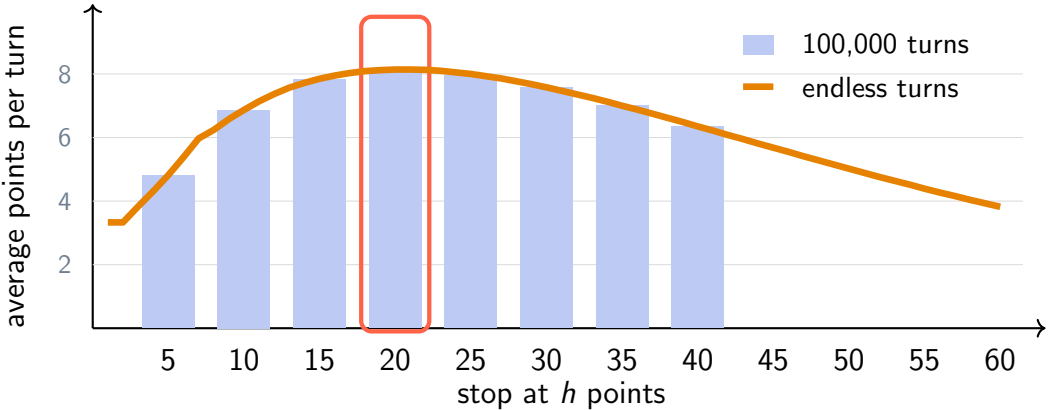
Try




Best: stop at **20**. Who guessed 20?

# Fixed: Points per Turn

Try




Endless turns: a smooth **hill**. Top: **20** (and 21), 8.14.

# Why 20? The Maths Agrees

You have $t$ points this turn. Roll **once more**?



1 time in 6:
**lose** $t$



5 times in 6:
**gain 4** (the average of 2 to 6)

# Why 20? The Maths Agrees

You have $t$ points this turn. Roll **once more**?



1 time in 6:
**lose** $t$



5 times in 6:
**gain 4** (the average of 2 to 6)

$$\text{Loss } \frac{t}{6} \quad \text{vs} \quad \text{gain } \frac{5 \times 4}{6} = \frac{20}{6}$$

# Why 20? The Maths Agrees

You have $t$ points this turn. Roll **once more**?



1 time in 6:
**lose** $t$

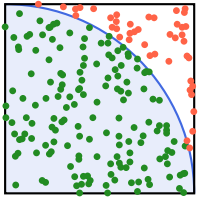


5 times in 6:
**gain 4** (the average of 2 to 6)

$$\text{Loss } \frac{t}{6} \quad \text{vs} \quad \text{gain } \frac{5 \times 4}{6} = \frac{20}{6}$$

Keep rolling while $t < 20$.

# Same Recipe, Other Questions



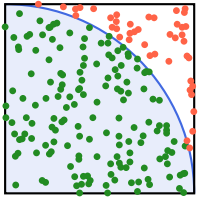
**Random darts**

$$4 \times \frac{157}{200} = \mathbf{3.14} \approx \pi$$

inside ÷ all ≈ quarter circle ÷ square

$= \pi/4$

# Same Recipe, Other Questions

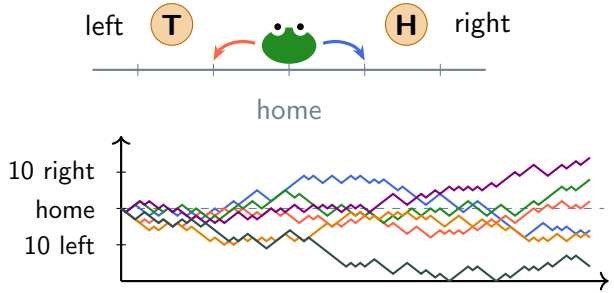


**Random darts**

$4 \times \frac{157}{200} = \mathbf{3.14} \approx \pi$

inside ÷ all ≈ quarter circle ÷ square

$= \pi/4$




6 frogs, 100 steps each

**A lost frog**

After 100 steps: about **8 steps** from home.

# Remember Hangman? (Assignment 2)

_ a _ _ e _ a secret word

**10** guesses

# Remember Hangman? (Assignment 2)

_ a _ _ e _ a secret word

**10** guesses

**right** letter:
it shows up

# Remember Hangman? (Assignment 2)

_ a _ _ e _ a secret word

**10** guesses

**right** letter: it shows up

**wrong** vowel: −2 guesses

**wrong** consonant: −1 guess

# Remember Hangman? (Assignment 2)

_ a _ _ e _ a secret word

**10** guesses

**right** letter: it shows up

**wrong** vowel: −2 guesses

**wrong** consonant: −1 guess

Fill every dash before the guesses run out.

# Remember Hangman? (Assignment 2)

_ a _ _ e _ a secret word

**10** guesses

**right** letter: it shows up

**wrong** vowel: −2 guesses

**wrong** consonant: −1 guess

Fill every dash before the guesses run out.

**Which letters should you guess first?**

# Two Computer Players

Each guesses letters in a **fixed order**.

Neither one looks at the dashes.



**random** order

d u r p a x e h ...

# Two Computer Players

Each guesses letters in a **fixed order**.

Neither one looks at the dashes.



**random** order

| d | u | r | p | a | x | e | h | ··· |
|---|---|---|---|---|---|---|---|---|



**most common** letters first

| e | s | i | a | r | n | t | o | ··· |
|---|---|---|---|---|---|---|---|---|

count every letter in A2's 55,900 words
(Al-Kindi's trick, Lecture 20)

# Two Computer Players

Each guesses letters in a **fixed order**.

Neither one looks at the dashes.



**random** order

| d | u | r | p | a | x | e | h | ··· |
|---|---|---|---|---|---|---|---|---|



**most common** letters first

| e | s | i | a | r | n | t | o | ··· |
|---|---|---|---|---|---|---|---|---|

count every letter in A2's 55,900 words (Al-Kindi's trick, Lecture 20)

Who wins more often? **How often?**

# One Game: `play`

```
1  def play(secret, order):
2      guesses = 10
3      guessed = []
4      for letter in order:
5          guessed.append(letter)
6          if letter not in secret:
7              if letter in 'aeiou':
8                  guesses -= 2
9              else:
10                 guesses -= 1
11         if has_player_won(secret, guessed):
12             return True
13         if guesses <= 0:
14             return False
```

`secret = 'planet'`
the hidden word

`order = 'esiarnt...'`
letters to try, in this order

`guessed`: letters tried so far

# One Game: `play`

```
1  def play(secret, order):
2      guesses = 10
3      guessed = []
4      for letter in order:
5          guessed.append(letter)
6          if letter not in secret:
7              if letter in 'aeiou':
8                  guesses -= 2
9              else:
10                 guesses -= 1
11         if has_player_won(secret, guessed):
12             return True
13         if guesses <= 0:
14             return False
```

`secret = 'planet'`
the hidden word

`order = 'esiarnt...'`
letters to try, in this order

`guessed`: letters tried so far

**Lines 4–5:** take the next letter of `order`.

# One Game: `play`

```
1  def play(secret, order):
2      guesses = 10
3      guessed = []
4      for letter in order:
5          guessed.append(letter)
6          if letter not in secret:
7              if letter in 'aeiou':
8                  guesses -= 2
9              else:
10                 guesses -= 1
11         if has_player_won(secret, guessed):
12             return True
13         if guesses <= 0:
14             return False
```



`secret = 'planet'`
the hidden word

`order = 'esiarnt...'`
letters to try, in this order

`guessed`: letters tried so far

**Lines 6–10:** not in the word? Vowel −2, else −1.

# One Game: `play`

```
1  def play(secret, order):
2      guesses = 10
3      guessed = []
4      for letter in order:
5          guessed.append(letter)
6          if letter not in secret:
7              if letter in 'aeiou':
8                  guesses -= 2
9              else:
10                 guesses -= 1
11         if has_player_won(secret, guessed):
12             return True
13         if guesses <= 0:
14             return False
```



`secret = 'planet'`
the hidden word

`order = 'esiarnt...'`
letters to try, in this order

`guessed`: letters tried so far

**Lines 11–12:**
word done? **Won**.
(`has_player_won`: yours, A2)

# One Game: `play`

```
1  def play(secret, order):
2      guesses = 10
3      guessed = []
4      for letter in order:
5          guessed.append(letter)
6          if letter not in secret:
7              if letter in 'aeiou':
8                  guesses -= 2
9              else:
10                 guesses -= 1
11         if has_player_won(secret, guessed):
12             return True
13         if guesses <= 0:
14             return False
```

`secret = 'planet'`
the hidden word

`order = 'esiarnt...'`
letters to try, in this order

`guessed`: letters tried so far

**Lines 13–14:** no guesses left? **Lost**.

# Many Games: the Same Recipe

the recipe

```
hits = 0
for i in range(n):
    play one game
    if it worked:
        hits += 1
print(hits / n)
```

for Hangman

```
def chance_to_win(order, n):
    wins = 0
    for i in range(n):
        word = random.choice(words)
        if play(word, order):
            wins += 1
    return wins / n
```

# Many Games: the Same Recipe

the recipe

```
hits = 0
for i in range(n):
    play one game
    if it worked:
        hits += 1
print(hits / n)
```

for Hangman

```
def chance_to_win(order, n):
    wins = 0
    for i in range(n):
        word = random.choice(words)
        if play(word, order):
            wins += 1
    return wins / n
```

Play one game: a **random** word, then `play`.

# Many Games: the Same Recipe

the recipe

```
hits = 0
for i in range(n):
    play one game
    if it worked:
        hits += 1
print(hits / n)
```

for Hangman

```
def chance_to_win(order, n):
    wins = 0
    for i in range(n):
        word = random.choice(words)
        if play(word, order):
            wins += 1
    return wins / n
```

It worked = `play` gave **True**: a win.

# Many Games: the Same Recipe

the recipe

```
hits = 0
for i in range(n):
    play one game
    if it worked:
        hits += 1
print(hits / n)
```

for Hangman

```
def chance_to_win(order, n):
    wins = 0
    for i in range(n):
        word = random.choice(words)
        if play(word, order):
            wins += 1
    return wins / n
```

Divide: wins ÷ games.

# Many Games: the Same Recipe

the recipe

```
hits = 0
for i in range(n):
    play one game
    if it worked:
        hits += 1
print(hits / n)
```

for Hangman

```
def chance_to_win(order, n):
    wins = 0
    for i in range(n):
        word = random.choice(words)
        if play(word, order):
            wins += 1
    return wins / n
```

`chance_to_win('esiarnt...', 100000)`: **22,223** wins.

22,223 ÷ 100,000 = **0.22**: the most-common player wins **22%**.

# Who Wins More?

100,000 games each.



random order



most common first

# Who Wins More?

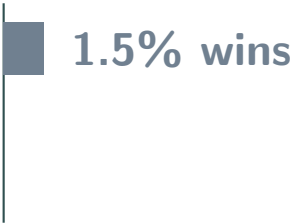
100,000 games each.



random order

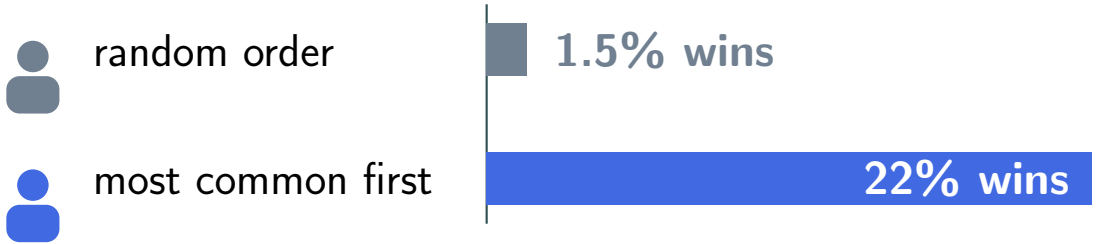


most common first



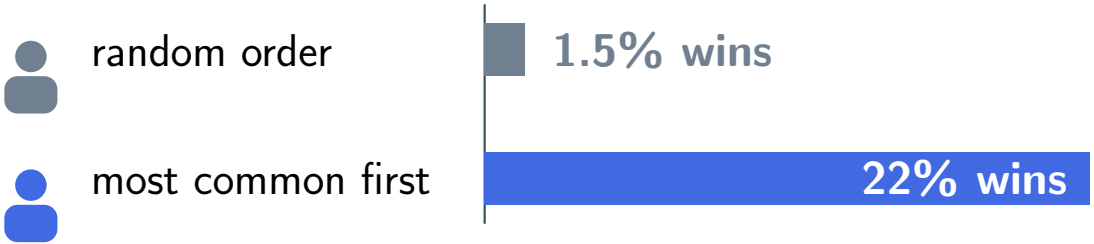
# Who Wins More?

100,000 games each.



# Who Wins More?

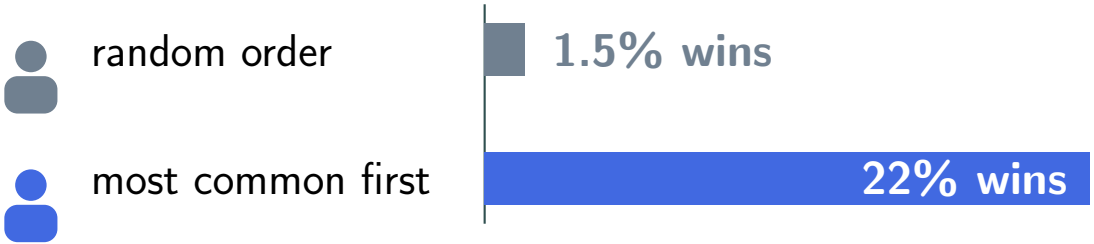
100,000 games each.




Counting letters: about **15**× more wins.

# Who Wins More?

100,000 games each.

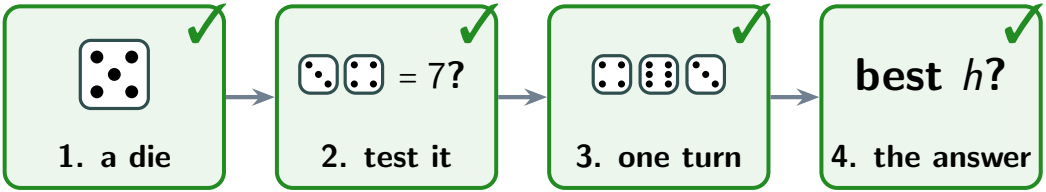



Counting letters: about **15**× more wins.

Real players also look at the dashes, so they do even better.

# Summary

# Summary




`randint(1, 6)` both ends in

100× tries → 1 more digit

a 1: the turn gives **0**

best $h$: 20

`random.seed(7)`: the same "random" numbers again.

# Big Idea

Ulam could not work out his chance to win.

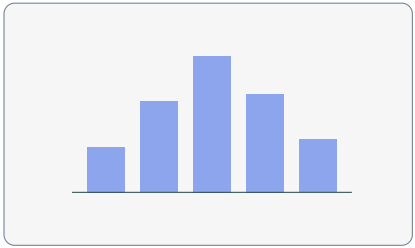
So he **played** and **counted**.

**Cannot solve it? Simulate it.**

Try many times. Count. Divide.

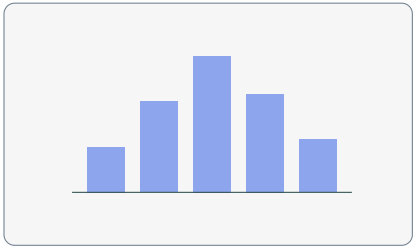
Best $h$ for points per turn: **20**. Did you guess it?

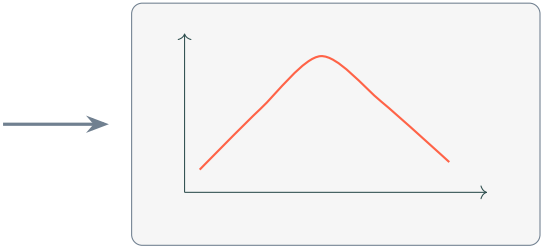
# Next Time: Plotting



today: drawn **by hand**

# Next Time: Plotting

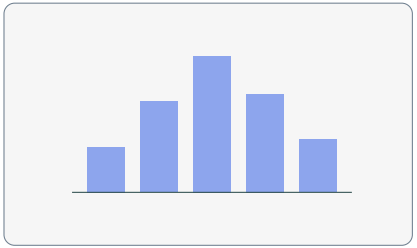




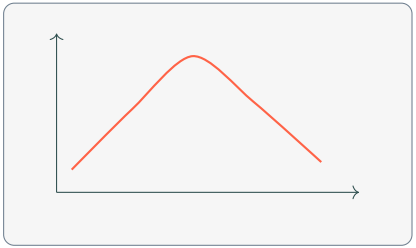
today: drawn **by hand**

`import matplotlib`

# Next Time: Plotting



today: drawn **by hand**



`import matplotlib`

London, 1854: a **map** of deaths finds the killer.

# Questions?

# Exit Ticket

## Today's question *(2 minutes)*

**(a)** Write **one** line that prints `ROLL` or `STOP`, each half the time.

**(b)** Ulam's 100 games gave his chance to about **1** digit. About how many games for **2** correct digits?

**COMP 102 — Exit Ticket #23**

Name: _____

Roll #: _____ Date: _____

**Answer:**

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