Today.

Finish undecidability.

Today.

Finish undecidability.

Start counting.

Halting Problem.

Halting Problem.

$HALT(P, I)$

Halting Problem.

$HALT(P, I)$

P - program

Halting Problem.

$HALT(P, I)$

P - program

I - input.

Halting Problem.

$HALT(P, I)$
P - program
I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Halting Problem.

$HALT(P, I)$
P - program
I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Notice:

Halting Problem.

$HALT(P, I)$
  P - program
  I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Notice:
Need a computer

Halting Problem.

$HALT(P, I)$
P - program
I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!

Halting Problem.

$HALT(P, I)$
  P - program
  I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine!

Halting Problem.

$HALT(P, I)$
P - program
I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Halting Problem.

$HALT(P, I)$

P - program

I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Halting Problem.

$HALT(P, I)$
P - program
I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Program is a text string.

Halting Problem.

$HALT(P, I)$
P - program
I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Program is a text string.
Text string can be an input to a program.

Halting Problem.

$HALT(P, I)$
 P - program
 I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Program is a text string.
Text string can be an input to a program.
Program can be an input to a program.

Halting Problem.

$HALT(P, I)$
P - program
I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Notice:
Need a computer
...with the notion of a stored program!!!!
(not an adding machine! not a person and an adding machine.)

Program is a text string.
Text string can be an input to a program.
Program can be an input to a program.

Implement HALT?

Implement HALT?

$HALT(P, I)$

Implement HALT?

$HALT(P, I)$

P - program

Implement HALT?

$HALT(P, I)$

P - program

I - input.

Implement HALT?

$HALT(P, I)$
P - program
I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Implement HALT?

$HALT(P, I)$
    P - program
    I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Run P on I and check!

Implement HALT?

$HALT(P, I)$

P - program

I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Run P on I and check!

How long do you wait?

Implement HALT?

$HALT(P, I)$
P - program
I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Run P on I and check!

How long do you wait?

Something about infinity here, maybe?

Implement HALT?

$HALT(P, I)$

P - program

I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Run P on I and check!

How long do you wait?

Something about infinity here, maybe?

Halt does not exist.

Halt does not exist.

$HALT(P, I)$

Halt does not exist.

$HALT(P, I)$

P - program

Halt does not exist.

$HALT(P, I)$

P - program

I - input.

Halt does not exist.

$HALT(P, I)$

  P - program

  I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Halt does not exist.

$HALT(P, I)$

P - program

I - input.

Determines if $P(I)$ (P run on I) halts or loops forever.

Theorem: There is no program HALT.

Halt and Turing.

Proof:

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot,\cdot)$.

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot,\cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot,\cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) =“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that “is” the program HALT.
There is text that is the program Turing.

Halt and Turing.

Proof: Assume there is a program $\mathit{HALT}(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot,\cdot)$.

Turing(P)
1. If HALT(P,P) =“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that “is” the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) =“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that “is” the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts
$\implies$ then HALTS(Turing, Turing) $=$ halts

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts
$\implies$ then HALTS(Turing, Turing) $=$ halts
$\implies$ Turing(Turing) loops forever.

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot,\cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts
$\implies$ then HALTS(Turing, Turing) $=$ halts
$\implies$ Turing(Turing) loops forever.

Case 2: Turing(Turing) loops forever

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot,\cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts
$\implies$ then HALTS(Turing, Turing) $=$ halts
$\implies$ Turing(Turing) loops forever.

Case 2: Turing(Turing) loops forever
$\implies$ then HALTS(Turing, Turing) $\neq$ halts

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot,\cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts
$\implies$ then HALTS(Turing, Turing) $=$ halts
$\implies$ Turing(Turing) loops forever.

Case 2: Turing(Turing) loops forever
$\implies$ then HALTS(Turing, Turing) $\neq$ halts
$\implies$ Turing(Turing) halts.

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot,\cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts
$\implies$ then HALTS(Turing, Turing) $=$ halts
$\implies$ Turing(Turing) loops forever.

Case 2: Turing(Turing) loops forever
$\implies$ then HALTS(Turing, Turing) $\neq$ halts
$\implies$ Turing(Turing) halts.

Contradiction.

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts
$\implies$ then HALTS(Turing, Turing) $=$ halts
$\implies$ Turing(Turing) loops forever.

Case 2: Turing(Turing) loops forever
$\implies$ then HALTS(Turing, Turing) $\neq$ halts
$\implies$ Turing(Turing) halts.

Contradiction. Program HALT does not exist!

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts
$\implies$ then HALTS(Turing, Turing) $=$ halts
$\implies$ Turing(Turing) loops forever.

Case 2: Turing(Turing) loops forever
$\implies$ then HALTS(Turing, Turing) $\neq$ halts
$\implies$ Turing(Turing) halts.

Contradiction. Program HALT does not exist! $\square$

Halt and Turing.

Proof: Assume there is a program $HALT(\cdot, \cdot)$.

Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Assumption: there is a program HALT.
There is text that "is" the program HALT.
There is text that is the program Turing. See above!
Can run Turing on Turing!

Does Turing(Turing) halt?

Case 1: Turing(Turing) halts
$\implies$ then HALTS(Turing, Turing) $=$ halts
$\implies$ Turing(Turing) loops forever.

Case 2: Turing(Turing) loops forever
$\implies$ then HALTS(Turing, Turing) $\neq$ halts
$\implies$ Turing(Turing) halts.

Contradiction. Program HALT does not exist! □
Questions?

Another view of proof: diagonalization.

Any program is a fixed length string.

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

	P_1	P_2	P_3	$\cdots$
P_1	H	H	L	$\cdots$
P_2	L	L	H	$\cdots$
P_3	L	H	H	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

	P_1	P_2	P_3	$\cdots$
P_1	H	H	L	$\cdots$
P_2	L	L	H	$\cdots$
P_3	L	H	H	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Halt - diagonal.

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

	P_1	P_2	P_3	$\cdots$
P_1	H	H	L	$\cdots$
P_2	L	L	H	$\cdots$
P_3	L	H	H	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Halt - diagonal.
Turing - is not Halt.

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

	P_1	P_2	P_3	$\cdots$
P_1	H	H	L	$\cdots$
P_2	L	L	H	$\cdots$
P_3	L	H	H	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Halt - diagonal.
Turing - is not Halt.
and is different from every P_i on the diagonal.

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

	P_1	P_2	P_3	$\cdots$
P_1	H	H	L	$\cdots$
P_2	L	L	H	$\cdots$
P_3	L	H	H	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Halt - diagonal.
Turing - is not Halt.
and is different from every P_i on the diagonal.
Turing is not on list.

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

	P_1	P_2	P_3	$\cdots$
P_1	H	H	L	$\cdots$
P_2	L	L	H	$\cdots$
P_3	L	H	H	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Halt - diagonal.
Turing - is not Halt.
and is different from every P_i on the diagonal.
Turing is not on list. Turing is not a program.

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

	P_1	P_2	P_3	$\cdots$
P_1	H	H	L	$\cdots$
P_2	L	L	H	$\cdots$
P_3	L	H	H	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Halt - diagonal.
Turing - is not Halt.
and is different from every P_i on the diagonal.
Turing is not on list. Turing is not a program.
Turing can be constructed from Halt.

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

	P_1	P_2	P_3	$\cdots$
P_1	H	H	L	$\cdots$
P_2	L	L	H	$\cdots$
P_3	L	H	H	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Halt - diagonal.
Turing - is not Halt.
and is different from every P_i on the diagonal.
Turing is not on list. Turing is not a program.
Turing can be constructed from Halt.
Halt does not exist!

Another view of proof: diagonalization.

Any program is a fixed length string.
Fixed length strings are enumerable.
Program halts or not any input, which is a string.

	P_1	P_2	P_3	$\cdots$
P_1	H	H	L	$\cdots$
P_2	L	L	H	$\cdots$
P_3	L	H	H	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Halt - diagonal.
Turing - is not Halt.
and is different from every P_i on the diagonal.
Turing is not on list. Turing is not a program.
Turing can be constructed from Halt.
Halt does not exist! □

Wow.

Proof play by play.

Assumed HALT(P, I) existed.

Proof play by play.

Assumed HALT(P, I) existed.

What is P?

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I?

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have ____ that is the program TURING.

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have Text that is the program TURING.

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have <u>Text</u> that is the program TURING.
Here it is!!

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have Text that is the program TURING.
Here it is!!
Turing(P)

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have Text that is the program TURING.
Here it is!!
Turing(P)
1. If HALT(P,P) =“halts”, then go into an infinite loop.

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have Text that is the program TURING.
Here it is!!
Turing(P)

1. If HALT(P,P) =“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have Text that is the program TURING.
Here it is!!
Turing(P)

1. If HALT(P,P) =“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Turing “diagonalizes” on list of program.

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have Text that is the program TURING.
Here it is!!
Turing(P)

1. If HALT(P,P) =“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Turing “diagonalizes” on list of program.

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have Text that is the program TURING.
Here it is!!
Turing(P)

1. If HALT(P,P) =“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Turing “diagonalizes” on list of program.
It is not a program!!!!

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have Text that is the program TURING.
Here it is!!
Turing(P)
1. If HALT(P,P) ="halts", then go into an infinite loop.
2. Otherwise, halt immediately.

Turing "diagonalizes" on list of program.
It is not a program!!!!
$\Longrightarrow$ HALT is not a program.

Proof play by play.

Assumed HALT(P, I) existed.

What is P? Text.
What is I? Text.

What does it mean to have a program HALT(P, I).
You have *Text* that is the program HALT(P, I).

Have Text that is the program TURING.
Here it is!!
Turing(P)

1. If HALT(P,P) =“halts”, then go into an infinite loop.
2. Otherwise, halt immediately.

Turing “diagonalizes” on list of program.
It is not a program!!!!
$\implies$ HALT is not a program.

Questions?

We are so smart!

Wow, that was easy!

We are so smart!

Wow, that was easy!

We should be famous!

No computers for Turing!

In Turing's time.

No computers for Turing!

In Turing's time.

No computers.

No computers for Turing!

In Turing’s time.

No computers.

Adding machines.

No computers for Turing!

In Turing's time.

No computers.

Adding machines.
e.g., Babbage (from table of logarithms) 1812.

No computers for Turing!

In Turing's time.

No computers.

Adding machines.
e.g., Babbage (from table of logarithms) 1812.

Concept of program as data wasn't really there.

Turing machine.

Turing machine.

A Turing machine.
– an (infinite) tape with characters

Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character

Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine

Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine

Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine
– where the tape could be a description of a ...

Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine
– where the tape could be a description of a ... Turing machine!

Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine
– where the tape could be a description of a ... Turing machine!

Now that's a computer!

Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine
– where the tape could be a description of a ... Turing machine!

Now that's a computer!

Turing: AI,

Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine
– where the tape could be a description of a ... Turing machine!

Now that's a computer!

Turing: AI, self modifying code,

Turing machine.

A Turing machine.
– an (infinite) tape with characters
– be in a state, and read a character
– move left, right, and/or write a character.

Universal Turing machine
– an interpreter program for a Turing machine
– where the tape could be a description of a ... Turing machine!

Now that's a computer!

Turing: AI, self modifying code, learning...

Turing and computing.

Just a mathematician?

Turing and computing.

Just a mathematician?

“Wrote” a chess program.

Turing and computing.

Just a mathematician?

“Wrote” a chess program.

Simulated the program by hand to play chess.

Turing and computing.

Just a mathematician?

“Wrote” a chess program.

Simulated the program by hand to play chess.

It won!

Turing and computing.

Just a mathematician?

“Wrote” a chess program.

Simulated the program by hand to play chess.

It won! Once anyway.

Turing and computing.

Just a mathematician?

“Wrote” a chess program.

Simulated the program by hand to play chess.

It won! Once anyway.

Involved with computing labs through the 40s.

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is...

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part.

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages!

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to
.... a natural number!

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to
.... a natural number! !

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to
.... a natural number! ! !

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to
.... a natural number! ! ! !

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to
.... a natural number! ! ! ! Same cardinality as...

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to
.... a natural number! ! ! ! Same cardinality as...Text.

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to
.... a natural number! ! ! ! Same cardinality as...Text.
Today:

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to
.... a natural number! ! ! ! Same cardinality as...Text.
Today:Programs can be written in

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to
.... a natural number! ! ! ! Same cardinality as...Text.
Today:Programs can be written in ascii.

Church, Gödel and Turing.

Church proved an equivalent theorem. (Previously.)

Used λ calculus....which is... Lisp (Scheme)!!!
.. functional part. Scheme's lambda is calculus's λ!

Programming languages! javascript, ruby, python....

Gödel: Incompleteness theorem.

Any formal system either is inconsistent or incomplete.
Inconsistent: A false sentence can be proven.
Incomplete: There is no proof for some sentence in the system.

Along the way: "built" computers out of arithmetic.
Showed that every mathematical statement corresponds to
.... a natural number! ! ! ! Same cardinality as...Text.
Today:Programs can be written in ascii.

Computing on top of computing...

Computer, assembly code, programming language, browser, html, javascript..

Computing on top of computing...

Computer, assembly code, programming language, browser, html, javascript..

We can't get enough of building more Turing machines.

Undecidable problems.

Does a program, P, print "Hello World"?

Undecidable problems.

Does a program, P, print "Hello World"?
How?

Undecidable problems.

Does a program, P, print "Hello World"?
How? What is P?

Undecidable problems.

Does a program, P, print “Hello World”?
How? What is P? Text!!!!!!

Undecidable problems.

Does a program, *P*, print "Hello World"?
How? What is *P*? Text!!!!!!

Undecidable problems.

Does a program, P, print “Hello World”?
How? What is P? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Undecidable problems.

Does a program, *P*, print “Hello World”?
How? What is *P*? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Undecidable problems.

Does a program, *P*, print “Hello World”?
How? What is *P*? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane?

Undecidable problems.

Does a program, *P*, print “Hello World”?
How? What is *P*? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Undecidable problems.

Does a program, *P*, print “Hello World”?
How? What is *P*? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Undecidable problems.

Does a program, P, print “Hello World”?
How? What is P? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?

Undecidable problems.

Does a program, P, print "Hello World"?
How? What is P? Text!!!!!!

Find exit points and add statement: **Print** "Hello World."

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: " $x^n + y^n = 1$?"

Undecidable problems.

Does a program, *P*, print "Hello World"?
How? What is *P*? Text!!!!!!

Find exit points and add statement: **Print** "Hello World."

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: " $x^n + y^n = 1$?"
Problem is undecidable.

Undecidable problems.

Does a program, P, print "Hello World"?
How? What is P? Text!!!!!!

Find exit points and add statement: **Print** "Hello World."

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: " $x^n + y^n = 1$?"
Problem is undecidable.

Be careful!

Undecidable problems.

Does a program, *P*, print "Hello World"?
How? What is *P*? Text!!!!!!

Find exit points and add statement: **Print** "Hello World."

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: " $x^n + y^n = 1$?"
Problem is undecidable.

Be careful!

Undecidable problems.

Does a program, *P*, print “Hello World”?
How? What is *P*? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: “ $x^n + y^n = 1$?”
Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$?

Undecidable problems.

Does a program, *P*, print "Hello World"?
How? What is *P*? Text!!!!!!

Find exit points and add statement: **Print** "Hello World."

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: " $x^n + y^n = 1$?"
Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$?
(Diophantine equation.)

Undecidable problems.

Does a program, P, print “Hello World”?
How? What is P? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: “ $x^n + y^n = 1$?”
Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$?
(Diophantine equation.)

The answer is yes or no.

Undecidable problems.

Does a program, P, print “Hello World”?
How? What is P? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: “ $x^n + y^n = 1$?”
Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$?
(Diophantine equation.)

The answer is yes or no. This “problem” is not undecidable.

Undecidable problems.

Does a program, P, print “Hello World”?
How? What is P? Text!!!!!!

Find exit points and add statement: **Print** “Hello World.”

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: “ $x^n + y^n = 1$?”
Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$?
(Diophantine equation.)

The answer is yes or no. This “problem” is not undecidable.

Undecidability for Diophantine set of equations

Undecidable problems.

Does a program, P, print "Hello World"?
How? What is P? Text!!!!!!

Find exit points and add statement: **Print** "Hello World."

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: " $x^n + y^n = 1$?"
Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$?
(Diophantine equation.)

The answer is yes or no. This "problem" is not undecidable.

Undecidability for Diophantine set of equations
$\implies$ no program can take any set of integer equations and

Undecidable problems.

Does a program, P, print "Hello World"?
How? What is P? Text!!!!!!

Find exit points and add statement: **Print** "Hello World."

Can a set of notched tiles tile the infinite plane?
Proof: simulate a computer. Halts if finite.

Does a set of integer equations have a solution?
Example: " $x^n + y^n = 1$?"
Problem is undecidable.

Be careful!

Is there an integer solution to $x^n + y^n = 1$?
(Diophantine equation.)

The answer is yes or no. This "problem" is not undecidable.

Undecidability for Diophantine set of equations
$\Longrightarrow$ no program can take any set of integer equations and always correctly output whether it has an integer solution.

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis:

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number.

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn't really work.

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number.
  Math 54 doesn't really work.

      Almost dependent matrices.

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number.
  Math 54 doesn't really work.

  Almost dependent matrices.
- Seminal paper in mathematical biology.

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number.
  Math 54 doesn't really work.

  Almost dependent matrices.
- Seminal paper in mathematical biology.

  Person:

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number.
  Math 54 doesn't really work.
    Almost dependent matrices.
- Seminal paper in mathematical biology.
  Person: embryo is blob.

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number.
  Math 54 doesn't really work.
  Almost dependent matrices.
- Seminal paper in mathematical biology.
  Person: embryo is blob. Legs,

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number.
  Math 54 doesn't really work.
    Almost dependent matrices.
- Seminal paper in mathematical biology.
  Person: embryo is blob. Legs, arms,

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn't really work.
  Almost dependent matrices.
- Seminal paper in mathematical biology.
  Person: embryo is blob. Legs, arms, head....

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn't really work.
  Almost dependent matrices.
- Seminal paper in mathematical biology.
  Person: embryo is blob. Legs, arms, head.... How?

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number.
  Math 54 doesn't really work.
  Almost dependent matrices.
- Seminal paper in mathematical biology.
  Person: embryo is blob. Legs, arms, head.... How?
  Fly:

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn't really work.
  Almost dependent matrices.
- Seminal paper in mathematical biology.
  Person: embryo is blob. Legs, arms, head.... How?
  Fly: blob.

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn't really work.
  Almost dependent matrices.
- Seminal paper in mathematical biology.
  Person: embryo is blob. Legs, arms, head.... How?
  Fly: blob. Torso becomes striped.

More about Alan Turing.

- Brilliant codebreaker during WWII, helped break German Enigma Code (which probably shortened war by 1 year).
- Seminal paper in numerical analysis: Condition number. Math 54 doesn't really work.
  Almost dependent matrices.
- Seminal paper in mathematical biology.
  Person: embryo is blob. Legs, arms, head.... How?
  Fly: blob. Torso becomes striped.
  Developed chemical reaction-diffusion networks that break symmetry.

Back to technical..

This statement is a lie.

Back to technical..

This statement is a lie. Neither true nor false!

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

def Turing(P):

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
  if Halts(P,P): while(true): pass
  else:
    return
```

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
  if Halts(P,P): while(true): pass
  else:
    return
```

...Text of Halt...

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
 if Halts(P,P): while(true): pass
 else:
  return
```

...Text of Halt...

Halt Progam $\Longrightarrow$ Turing Program.

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
 if Halts(P,P): while(true): pass
 else:
  return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
  if Halts(P,P): while(true): pass
  else:
    return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
 if Halts(P,P): while(true): pass
 else:
   return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Turing("Turing")?

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
 if Halts(P,P): while(true): pass
 else:
  return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Turing("Turing")? Neither halts nor loops!

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
 if Halts(P,P): while(true): pass
 else:
  return
```

...Text of Halt...

Halt Progam $\Longrightarrow$ Turing Program. ($P \Longrightarrow Q$)

Turing("Turing")? Neither halts nor loops! $\Longrightarrow$ No Turing program.

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
  if Halts(P,P): while(true): pass
  else:
    return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Turing("Turing")? Neither halts nor loops! $\implies$ No Turing program.

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
  if Halts(P,P): while(true): pass
  else:
    return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Turing("Turing")? Neither halts nor loops! $\implies$ No Turing program.

No Turing Program

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn’t shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
  if Halts(P,P): while(true): pass
  else:
    return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Turing(“Turing”)? Neither halts nor loops! $\implies$ No Turing program.

No Turing Program $\implies$

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
  if Halts(P,P): while(true): pass
  else:
    return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Turing("Turing")? Neither halts nor loops! $\implies$ No Turing program.

No Turing Program $\implies$ No halt program.

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
  if Halts(P,P): while(true): pass
  else:
    return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Turing("Turing")? Neither halts nor loops! $\implies$ No Turing program.

No Turing Program $\implies$ No halt program. ($\neg Q \implies \neg P$)

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
  if Halts(P,P): while(true): pass
  else:
    return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Turing("Turing")? Neither halts nor loops! $\implies$ No Turing program.

No Turing Program $\implies$ No halt program. ($\neg Q \implies \neg P$)

Program is text, so we can pass it to itself,

Back to technical..

This statement is a lie. Neither true nor false!

Every person who doesn't shave themselves is shaved by the barber.

Who shaves the barber?

```
def Turing(P):
 if Halts(P,P): while(true): pass
 else:
   return
```

...Text of Halt...

Halt Progam $\implies$ Turing Program. ($P \implies Q$)

Turing("Turing")? Neither halts nor loops! $\implies$ No Turing program.

No Turing Program $\implies$ No halt program. ($\neg Q \implies \neg P$)

Program is text, so we can pass it to itself, or refer to self.

Summary: decidability.

Computer Programs are an interesting thing.

Summary: decidability.

Computer Programs are an interesting thing.
  Like Math.

Summary: decidability.

Computer Programs are an interesting thing.
  Like Math.
  Formal Systems.

Summary: decidability.

Computer Programs are an interesting thing.
  Like Math.
  Formal Systems.

Summary: decidability.

Computer Programs are an interesting thing.
  Like Math.
  Formal Systems.

Computer Programs cannot completely “understand” computer programs.

Summary: decidability.

Computer Programs are an interesting thing.
  Like Math.
  Formal Systems.

Computer Programs cannot completely “understand” computer programs.

Summary: decidability.

Computer Programs are an interesting thing.
  Like Math.
  Formal Systems.

Computer Programs cannot completely "understand" computer programs.

Computation is a lens for other action in the world.

Probability

What's to come?

Probability

What's to come? Probability.

Probability

What's to come? Probability.

A bag contains:

Probability

What's to come? Probability.

A bag contains:

Probability

What's to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?

Probability

What's to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?

Count blue.

Probability

What's to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?

Count blue. Count total.

Probability

What's to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?

Count blue. Count total. Divide.

Probability

What's to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?

Count blue. Count total. Divide.

For now:

Probability

What's to come? Probability.

A bag contains:

What is the chance that a ball taken from the bag is blue?

Count blue. Count total. Divide.

For now: Counting!

Outline: basics

1. Counting.
2. Tree
3. Rules of Counting
4. Sample with/without replacement where order does/doesn’t matter.

Count?

How many outcomes possible for k coin tosses?
How many poker hands?
How many handshakes for n people?
How many diagonals in a convex polygon?
How many 10 digit numbers?
How many 10 digit numbers without repetition?

Using a tree..

How many 3-bit strings?

Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

How would you make one sequence?

Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

How would you make one sequence?

How many different ways to do that making?

Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

How would you make one sequence?

How many different ways to do that making?

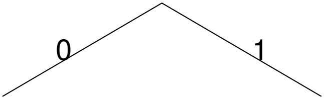
Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

How would you make one sequence?

How many different ways to do that making?



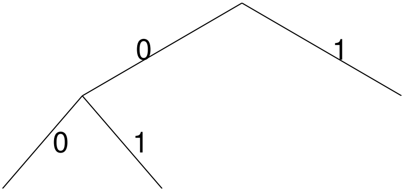
Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

How would you make one sequence?

How many different ways to do that making?



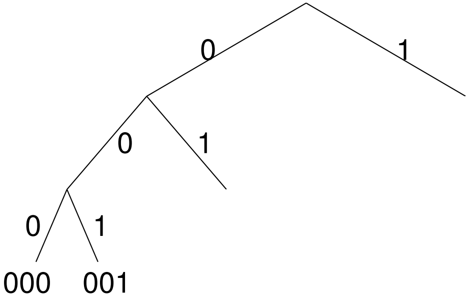
Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

How would you make one sequence?

How many different ways to do that making?



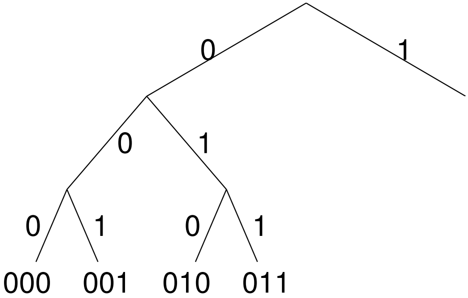
Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

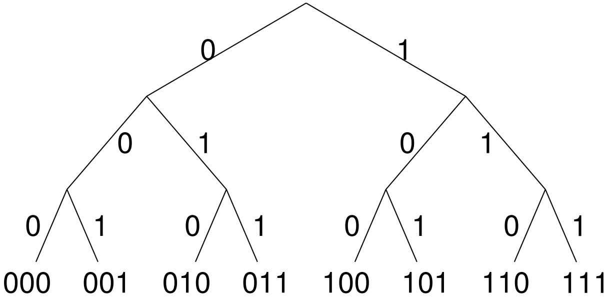
How would you make one sequence?

How many different ways to do that making?



Using a tree..

How many 3-bit strings?
How many different sequences of three bits from $\{0, 1\}$?
How would you make one sequence?
How many different ways to do that making?



8 leaves which is $2 \times 2 \times 2$.

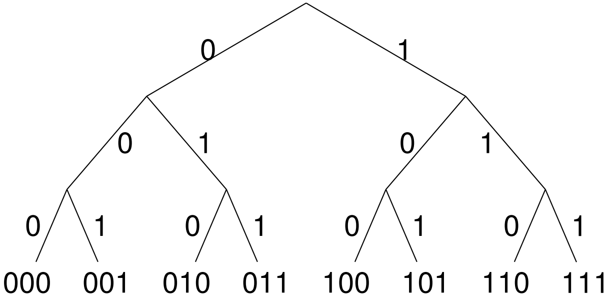
Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

How would you make one sequence?

How many different ways to do that making?



8 leaves which is $2 \times 2 \times 2$. One leaf for each string.

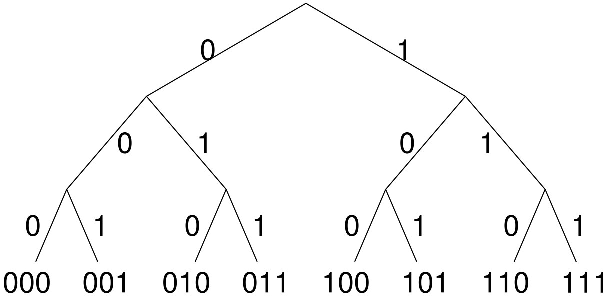
Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

How would you make one sequence?

How many different ways to do that making?



8 leaves which is $2 \times 2 \times 2$. One leaf for each string.

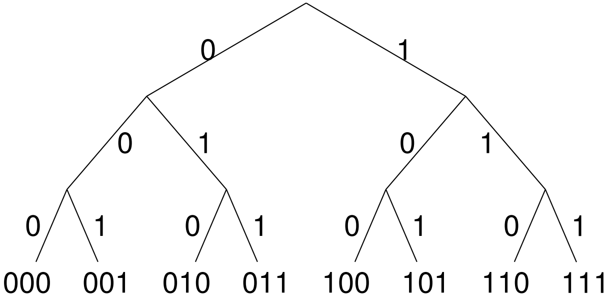
Using a tree..

How many 3-bit strings?

How many different sequences of three bits from $\{0, 1\}$?

How would you make one sequence?

How many different ways to do that making?



8 leaves which is $2 \times 2 \times 2$. One leaf for each string.

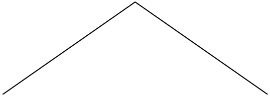
8 3-bit srings!

First Rule of Counting: Product Rule

Objects made by choosing from n_1, then n_2, ..., then n_k
the number of objects is $n_1 \times n_2 \cdots \times n_k$.

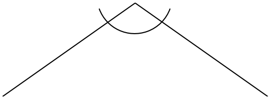
First Rule of Counting: Product Rule

Objects made by choosing from n_1, then n_2, ..., then n_k
the number of objects is $n_1 \times n_2 \cdots \times n_k$.



First Rule of Counting: Product Rule

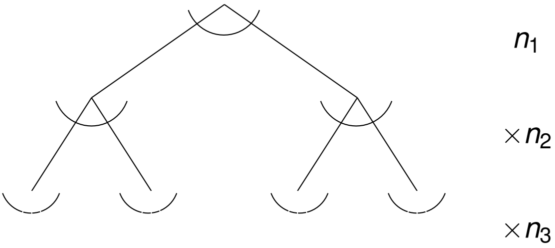
Objects made by choosing from n_1, then n_2, ..., then n_k
the number of objects is $n_1 \times n_2 \cdots \times n_k$.



n_1

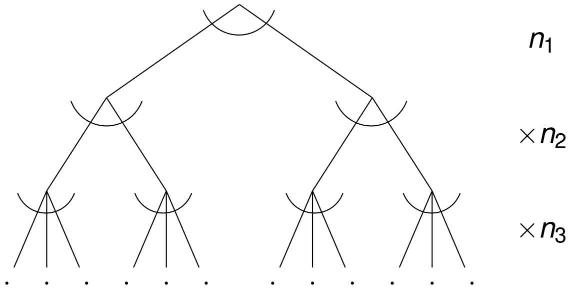
First Rule of Counting: Product Rule

Objects made by choosing from n_1, then n_2, ..., then n_k
the number of objects is $n_1 \times n_2 \cdots \times n_k$.



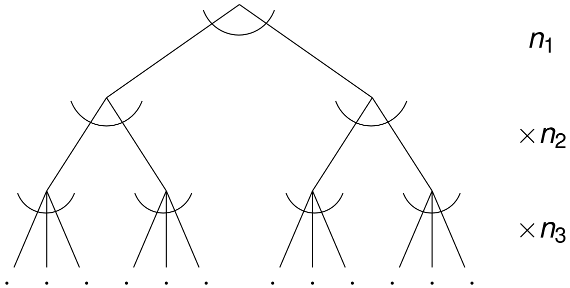
First Rule of Counting: Product Rule

Objects made by choosing from n_1, then n_2, …, then n_k
the number of objects is $n_1 \times n_2 \cdots \times n_k$.



First Rule of Counting: Product Rule

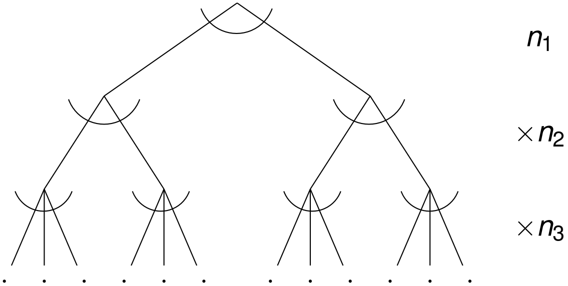
Objects made by choosing from n_1, then n_2, ..., then n_k the number of objects is $n_1 \times n_2 \cdots \times n_k$.



In picture, $2 \times 2 \times 3 = 12$!

First Rule of Counting: Product Rule

Objects made by choosing from n_1, then n_2, …, then n_k the number of objects is $n_1 \times n_2 \cdots \times n_k$.



In picture, $2 \times 2 \times 3 = 12$!

Using the first rule..

How many outcomes possible for k coin tosses?

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice,

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
2

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
2×2

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots$

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2$

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice,

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice, 10 ways for second choice, ...

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice, 10 ways for second choice, ...
10

Using the first rule..

How many outcomes possible for *k* coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice, 10 ways for second choice, ...
$10 \times$

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice, 10 ways for second choice, ...
$10 \times 10 \cdots$

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice, 10 ways for second choice, ...
$10 \times 10 \cdots \times 10$

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice, 10 ways for second choice, ...
$10 \times 10 \cdots \times 10 = 10^k$

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice, 10 ways for second choice, ...
$10 \times 10 \cdots \times 10 = 10^k$

How many n digit base m numbers?

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice, 10 ways for second choice, ...
$10 \times 10 \cdots \times 10 = 10^k$

How many n digit base m numbers?

m ways for first,

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice, 10 ways for second choice, ...
$10 \times 10 \cdots \times 10 = 10^k$

How many n digit base m numbers?

m ways for first, m ways for second, ...

Using the first rule..

How many outcomes possible for k coin tosses?

2 ways for first choice, 2 ways for second choice, ...
$2 \times 2 \cdots \times 2 = 2^k$

How many 10 digit numbers?

10 ways for first choice, 10 ways for second choice, ...
$10 \times 10 \cdots \times 10 = 10^k$

How many n digit base m numbers?

m ways for first, m ways for second, ...
m^n

Functions, polynomials.

How many functions f mapping S to T?

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$,

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$, $|T|$ ways to choose for $f(s_2)$, ...

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$, $|T|$ ways to choose for $f(s_2)$, ...

....$|T|^{|S|}$

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$, $|T|$ ways to choose for $f(s_2)$, ...

....$|T|^{|S|}$

How many polynomials of degree d modulo p?

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$, $|T|$ ways to choose for $f(s_2)$, ...

....$|T|^{|S|}$

How many polynomials of degree d modulo p?

p ways to choose for first coefficient,

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$, $|T|$ ways to choose for $f(s_2)$, ...

....$|T|^{|S|}$

How many polynomials of degree d modulo p?

p ways to choose for first coefficient, p ways for second, ...

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$, $|T|$ ways to choose for $f(s_2)$, ...

....$|T|^{|S|}$

How many polynomials of degree d modulo p?

p ways to choose for first coefficient, p ways for second, ...
...p^{d+1}

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$, $|T|$ ways to choose for $f(s_2)$, ...

....$|T|^{|S|}$

How many polynomials of degree d modulo p?

p ways to choose for first coefficient, p ways for second, ...
...p^{d+1}

p values for first point,

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$, $|T|$ ways to choose for $f(s_2)$, ...

....$|T|^{|S|}$

How many polynomials of degree d modulo p?

p ways to choose for first coefficient, p ways for second, ...
...p^{d+1}

p values for first point, p values for second, ...

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$, $|T|$ ways to choose for $f(s_2)$, ...

....$|T|^{|S|}$

How many polynomials of degree d modulo p?

p ways to choose for first coefficient, p ways for second, ...
...p^{d+1}

p values for first point, p values for second, ...
...p^{d+1}

Functions, polynomials.

How many functions f mapping S to T?

$|T|$ ways to choose for $f(s_1)$, $|T|$ ways to choose for $f(s_2)$, ...

....$|T|^{|S|}$

How many polynomials of degree d modulo p?

p ways to choose for first coefficient, p ways for second, ...
...p^{d+1}

p values for first point, p values for second, ...
...p^{d+1}

Questions?

Permutations.

[1] By definition: $0! = 1$.

Permutations.

How many 10 digit numbers **without repeating a digit**?

[1] By definition: $0! = 1$.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

[1] By definition: $0! = 1$.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first,

[1] By definition: $0! = 1$.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second,

[1] By definition: $0! = 1$.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third,

[1] By definition: $0! = 1$.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

[1] By definition: $0! = 1$.

Permutations.

How many 10 digit numbers **without repeating a digit**? (leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

[1] By definition: 0! = 1.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

[1] By definition: 0! = 1.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice,

[1] By definition: 0! = 1.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice, $n-1$ ways for second,

[1] By definition: 0! = 1.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice, $n-1$ ways for second,
$n-2$ choices for third,

[1] By definition: 0! = 1.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice, $n-1$ ways for second,
$n-2$ choices for third, ...

[1] By definition: 0! = 1.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice, $n-1$ ways for second,
$n-2$ choices for third, ...

... $n * (n-1) * (n-2) \cdot * (n-k+1) = \frac{n!}{(n-k)!}$.

[1] By definition: 0! = 1.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10*9*8\cdots*1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice, $n-1$ ways for second,
$n-2$ choices for third, ...

... $n*(n-1)*(n-2)\cdot*(n-k+1) = \frac{n!}{(n-k)!}$.

How many orderings of n objects are there?
Permutations of n objects.

[1] By definition: 0! = 1.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice, $n-1$ ways for second,
$n-2$ choices for third, ...

... $n*(n-1)*(n-2) \cdot *(n-k+1) = \frac{n!}{(n-k)!}$.

How many orderings of n objects are there?
Permutations of n objects.

n ways for first,

[1] By definition: 0! = 1.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice, $n-1$ ways for second,
$n-2$ choices for third, ...

... $n * (n-1) * (n-2) \cdot * (n-k+1) = \frac{n!}{(n-k)!}$.

How many orderings of n objects are there?
Permutations of *n* objects.

n ways for first, $n-1$ ways for second,

[1] By definition: $0! = 1$.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice, $n-1$ ways for second,
$n-2$ choices for third, ...

... $n*(n-1)*(n-2) \cdot *(n-k+1) = \frac{n!}{(n-k)!}$.

How many orderings of n objects are there?
Permutations of n objects.

n ways for first, $n-1$ ways for second,
$n-2$ ways for third,

[1] By definition: $0! = 1$.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10*9*8\cdots*1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice, $n-1$ ways for second,
$n-2$ choices for third, ...

... $n*(n-1)*(n-2)\cdot*(n-k+1) = \frac{n!}{(n-k)!}$.

How many orderings of n objects are there?
Permutations of n objects.

n ways for first, $n-1$ ways for second,
$n-2$ ways for third, ...

[1] By definition: 0! = 1.

Permutations.

How many 10 digit numbers **without repeating a digit**?
(leading zeros are ok.)

10 ways for first, 9 ways for second, 8 ways for third, ...

... $10 * 9 * 8 \cdots * 1 = 10!$.[1]

How many different samples of size k from n numbers **without replacement.**

n ways for first choice, $n-1$ ways for second,
$n-2$ choices for third, ...

... $n*(n-1)*(n-2) \cdot *(n-k+1) = \frac{n!}{(n-k)!}$.

How many orderings of n objects are there?
Permutations of n objects.

n ways for first, $n-1$ ways for second,
$n-2$ ways for third, ...

... $n*(n-1)*(n-2) \cdot *1 = n!$.

[1] By definition: 0! = 1.

One-to-One Functions.

One-to-One Functions.

How many one-to-one functions from $|S|$ to $|S|$.

One-to-One Functions.

How many one-to-one functions from $|S|$ to $|S|$.

$|S|$ choices for $f(s_1)$,

One-to-One Functions.

How many one-to-one functions from $|S|$ to $|S|$.

$|S|$ choices for $f(s_1)$, $|S|-1$ choices for $f(s_2)$, ...

One-to-One Functions.

How many one-to-one functions from $|S|$ to $|S|$.

$|S|$ choices for $f(s_1)$, $|S|-1$ choices for $f(s_2)$, ...

One-to-One Functions.

How many one-to-one functions from $|S|$ to $|S|$.

$|S|$ choices for $f(s_1)$, $|S|-1$ choices for $f(s_2)$, ...

So total number is $|S| \times |S|-1 \cdots 1 = |S|!$

One-to-One Functions.

How many one-to-one functions from $|S|$ to $|S|$.

$|S|$ choices for $f(s_1)$, $|S|-1$ choices for $f(s_2)$, ...

So total number is $|S| \times |S|-1 \cdots 1 = |S|!$
A one-to-one function is a permutation!

Counting sets..when order doesn't matter.

How many poker hands?

[2] When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$

[2]When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$???

---

[2] When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$???

Are $A, K, Q, 10, J$ of spades

[2] When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$???

Are $A, K, Q, 10, J$ of spades
and $10, J, Q, K, A$ of spades

[2]When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$???

Are $A, K, Q, 10, J$ of spades
and $10, J, Q, K, A$ of spades the same?

[2]When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$???

Are $A, K, Q, 10, J$ of spades
and $10, J, Q, K, A$ of spades the same?

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.[2]

[2]When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$???

Are $A, K, Q, 10, J$ of spades
and $10, J, Q, K, A$ of spades the same?

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.[2]

Number of orderings for a poker hand: "5!"

[2]When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$???

Are $A, K, Q, 10, J$ of spades
and $10, J, Q, K, A$ of spades the same?

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.[2]

Number of orderings for a poker hand: "5!"
(The "!" means factorial, not Exclamation.)

---

[2]When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$???

Are $A, K, Q, 10, J$ of spades
and $10, J, Q, K, A$ of spades the same?

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.[2]

Number of orderings for a poker hand: "5!"

$$\frac{52 \times 51 \times 50 \times 49 \times 48}{5!}$$

[2] When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$???

Are $A, K, Q, 10, J$ of spades
and $10, J, Q, K, A$ of spades the same?

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.[2]

Number of orderings for a poker hand: "5!"

$$\frac{52 \times 51 \times 50 \times 49 \times 48}{5!}$$

Can write as...

$$\frac{52!}{5! \times 47!}$$

[2] When each unordered object corresponds equal numbers of ordered objects.

Counting sets..when order doesn't matter.

How many poker hands?

$52 \times 51 \times 50 \times 49 \times 48$???

Are $A, K, Q, 10, J$ of spades
and $10, J, Q, K, A$ of spades the same?

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.[2]

Number of orderings for a poker hand: "5!"

Can write as...

$$\frac{52 \times 51 \times 50 \times 49 \times 48}{5!}$$

$$\frac{52!}{5! \times 47!}$$

Generic: ways to choose 5 out of 52 possibilities.

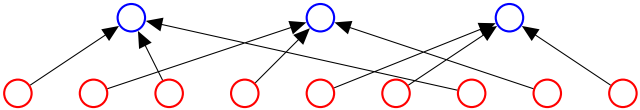
[2] When each unordered object corresponds equal numbers of ordered objects.

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.

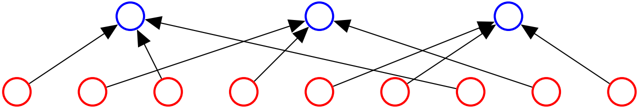
Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



Ordered to unordered.

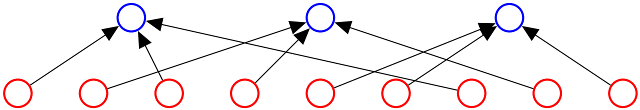
Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



How many red nodes (ordered objects)?

Ordered to unordered.

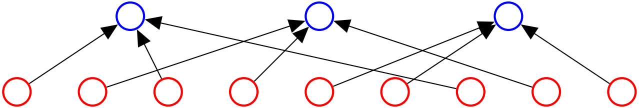
Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



How many red nodes (ordered objects)? 9.

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.

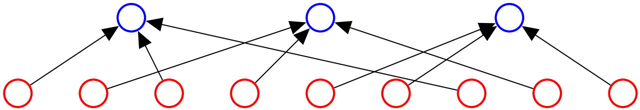


How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node?

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.

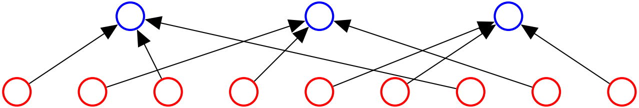


How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node? 3.

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



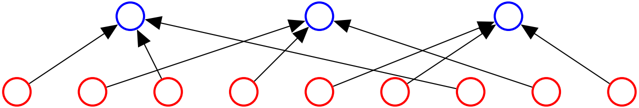
How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node? 3.

How many blue nodes (unordered objects)?

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



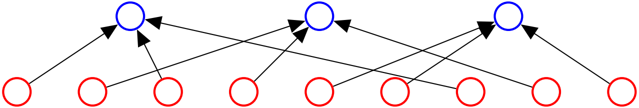
How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node? 3.

How many blue nodes (unordered objects)? $\frac{9}{3}$

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



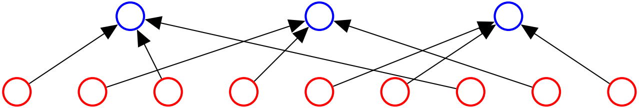
How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node? 3.

How many blue nodes (unordered objects)? $\frac{9}{3} = 3$.

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



How many red nodes (ordered objects)? 9.

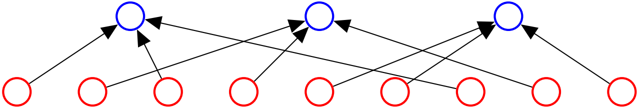
How many red nodes mapped to one blue node? 3.

How many blue nodes (unordered objects)? $\frac{9}{3} = 3$.

How many poker deals?

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



How many red nodes (ordered objects)? 9.

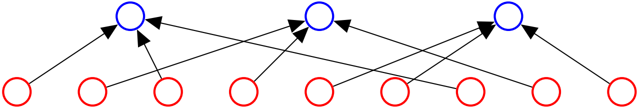
How many red nodes mapped to one blue node? 3.

How many blue nodes (unordered objects)? $\frac{9}{3} = 3$.

How many poker deals? $52 \cdot 51 \cdot 50 \cdot 49 \cdot 48$.

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node? 3.

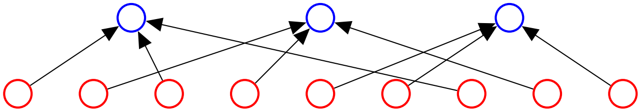
How many blue nodes (unordered objects)? $\frac{9}{3} = 3$.

How many poker deals? $52 \cdot 51 \cdot 50 \cdot 49 \cdot 48$.

How many poker deals per hand?

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node? 3.

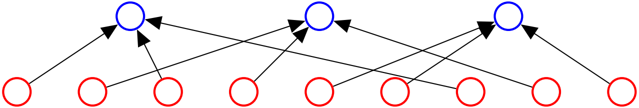
How many blue nodes (unordered objects)? $\frac{9}{3} = 3$.

How many poker deals? $52 \cdot 51 \cdot 50 \cdot 49 \cdot 48$.

How many poker deals per hand?
Map each deal to ordered deal:

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node? 3.

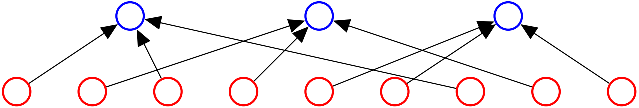
How many blue nodes (unordered objects)? $\frac{9}{3} = 3$.

How many poker deals? $52 \cdot 51 \cdot 50 \cdot 49 \cdot 48$.

How many poker deals per hand?
Map each deal to ordered deal: 5!

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node? 3.

How many blue nodes (unordered objects)? $\frac{9}{3} = 3$.

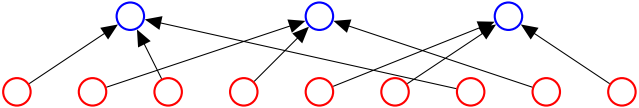
How many poker deals? $52 \cdot 51 \cdot 50 \cdot 49 \cdot 48$.

How many poker deals per hand?
  Map each deal to ordered deal: 5!

How many poker hands?

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node? 3.

How many blue nodes (unordered objects)? $\frac{9}{3} = 3$.

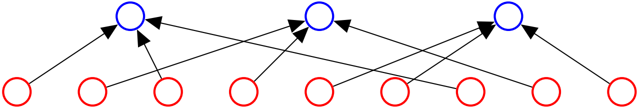
How many poker deals? $52 \cdot 51 \cdot 50 \cdot 49 \cdot 48$.

How many poker deals per hand?
Map each deal to ordered deal: 5!

How many poker hands? $\frac{52 \cdot 51 \cdot 50 \cdot 49 \cdot 48}{5!}$

Ordered to unordered.

Second Rule of Counting: If order doesn't matter count ordered objects and then divide by number of orderings.



How many red nodes (ordered objects)? 9.

How many red nodes mapped to one blue node? 3.

How many blue nodes (unordered objects)? $\frac{9}{3} = 3$.

How many poker deals? $52 \cdot 51 \cdot 50 \cdot 49 \cdot 48$.

How many poker deals per hand?
   Map each deal to ordered deal: 5!

How many poker hands? $\frac{52 \cdot 51 \cdot 50 \cdot 49 \cdot 48}{5!}$

Questions?

..order doesn’t matter.

..order doesn't matter.

Choose 2 out of n?

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{}$$

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2}$$

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

Choose 3 out of n?

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

Choose 3 out of n?

$$\frac{n \times (n-1) \times (n-2)}{}$$

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

Choose 3 out of n?

$$\frac{n \times (n-1) \times (n-2)}{3!}$$

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

Choose 3 out of n?

$$\frac{n \times (n-1) \times (n-2)}{3!} = \frac{n!}{(n-3)! \times 3!}$$

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

Choose 3 out of n?

$$\frac{n \times (n-1) \times (n-2)}{3!} = \frac{n!}{(n-3)! \times 3!}$$

Choose k **out of** n?

$$\frac{n!}{(n-k)!}$$

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

Choose 3 out of n?

$$\frac{n \times (n-1) \times (n-2)}{3!} = \frac{n!}{(n-3)! \times 3!}$$

Choose k **out of** n?

$$\frac{n!}{(n-k)!\qquad\qquad}$$

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

Choose 3 out of n?

$$\frac{n \times (n-1) \times (n-2)}{3!} = \frac{n!}{(n-3)! \times 3!}$$

Choose k **out of** n?

$$\frac{n!}{(n-k)! \times k!}$$

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

Choose 3 out of n?

$$\frac{n \times (n-1) \times (n-2)}{3!} = \frac{n!}{(n-3)! \times 3!}$$

Choose k **out of** n?

$$\frac{n!}{(n-k)! \times k!}$$

Notation: $\binom{n}{k}$ **and pronounced "n choose k."**

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

Choose 3 out of n?

$$\frac{n \times (n-1) \times (n-2)}{3!} = \frac{n!}{(n-3)! \times 3!}$$

Choose k **out of** n?

$$\frac{n!}{(n-k)! \times k!}$$

Notation: $\binom{n}{k}$ **and pronounced "**n **choose** k**."**

Familiar?

..order doesn't matter.

Choose 2 out of n?

$$\frac{n \times (n-1)}{2} = \frac{n!}{(n-2)! \times 2}$$

Choose 3 out of n?

$$\frac{n \times (n-1) \times (n-2)}{3!} = \frac{n!}{(n-3)! \times 3!}$$

Choose k **out of** n?

$$\frac{n!}{(n-k)! \times k!}$$

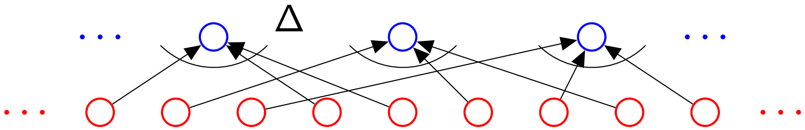
Notation: $\binom{n}{k}$ **and pronounced "n choose k."**

Familiar? Questions?

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**

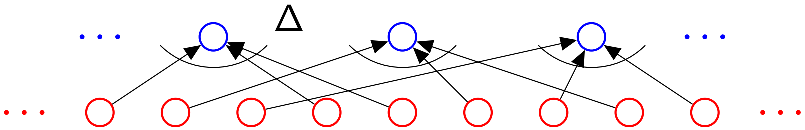
Second rule: when order doesn't matter divide...



Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**

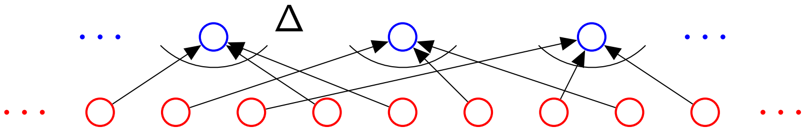
Second rule: when order doesn't matter divide...



3 card Poker deals: 52

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...

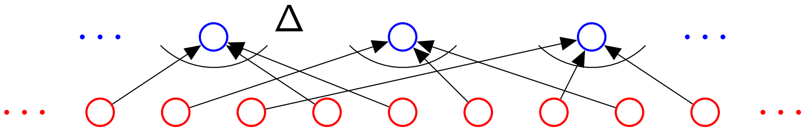


3 card Poker deals: 52×51

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**

Second rule: when order doesn't matter divide...

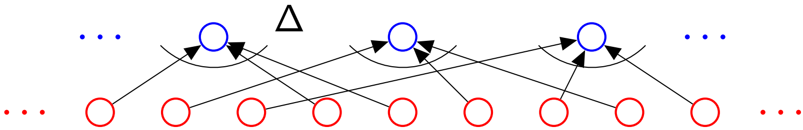


3 card Poker deals: $52 \times 51 \times 50$

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**

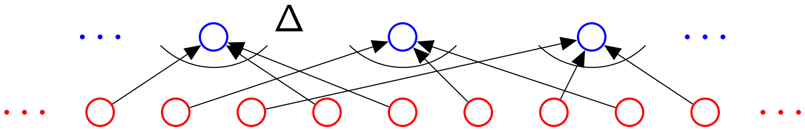
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$.

Example: Visualize the proof..

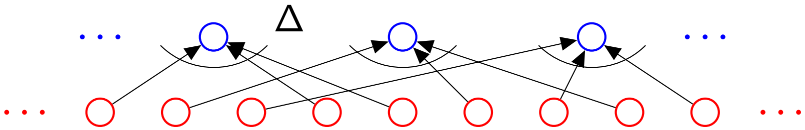
First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.

Example: Visualize the proof..

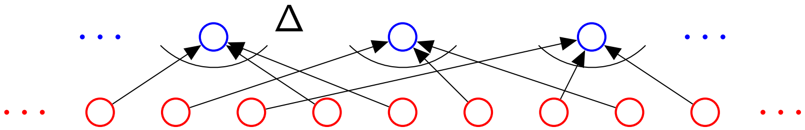
First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?

Example: Visualize the proof..

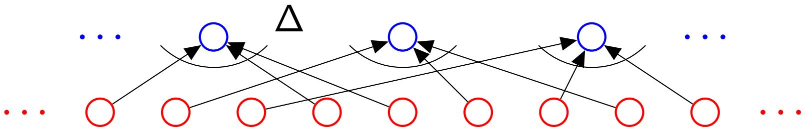
First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?
Hand: Q, K, A.

Example: Visualize the proof..

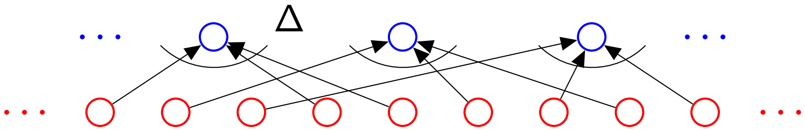
First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?
Hand: Q, K, A.
Deals: Q, K, A :

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



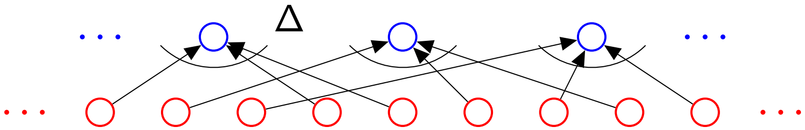
3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?

Hand: Q, K, A.

Deals: Q, K, A : Q, A, K :

Example: Visualize the proof..

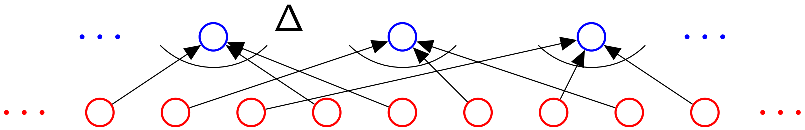
First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?
Hand: Q, K, A.
Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?

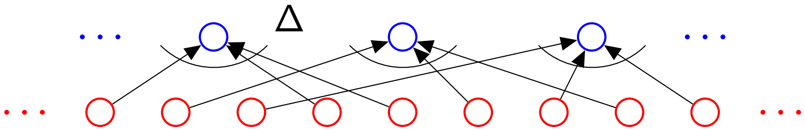
Hand: Q, K, A.

Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.

$\Delta = 3 \times 2 \times 1$

Example: Visualize the proof..

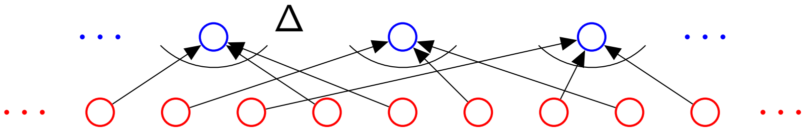
First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?
Hand: Q, K, A.
Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.
$\Delta = 3 \times 2 \times 1$ First rule again.

Example: Visualize the proof..

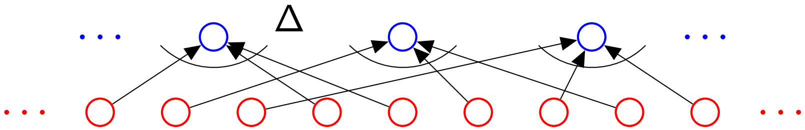
First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?
Hand: Q, K, A.
Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.
$\Delta = 3 \times 2 \times 1$ First rule again.
Total:

Example: Visualize the proof..

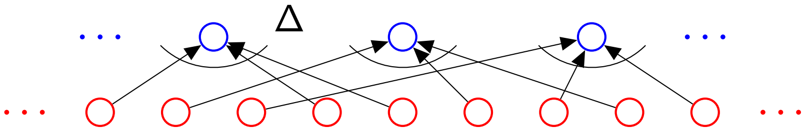
First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?
Hand: Q, K, A.
Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.
$\Delta = 3 \times 2 \times 1$ First rule again.
Total: $\frac{52!}{49!3!}$

Example: Visualize the proof..

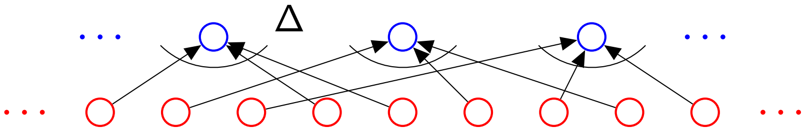
First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?
Hand: Q, K, A.
Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.
$\Delta = 3 \times 2 \times 1$ First rule again.
Total: $\frac{52!}{49!3!}$ Second Rule!

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...

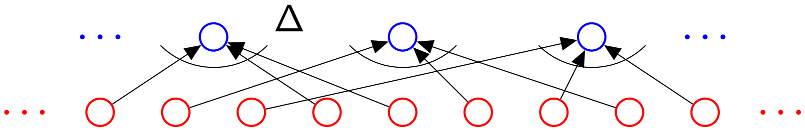


3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?
Hand: Q, K, A.
Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.
$\Delta = 3 \times 2 \times 1$ First rule again.
Total: $\frac{52!}{49!3!}$ Second Rule!

Choose k out of n.

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?

Hand: Q, K, A.

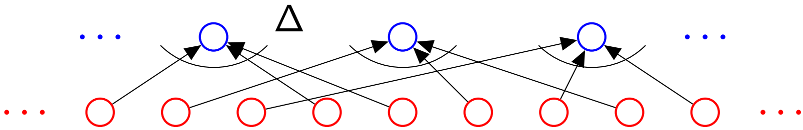
Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.

$\Delta = 3 \times 2 \times 1$ First rule again.
Total: $\frac{52!}{49!3!}$ Second Rule!

Choose k out of n.
Ordered set: $\frac{n!}{(n-k)!}$

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?

Hand: Q, K, A.

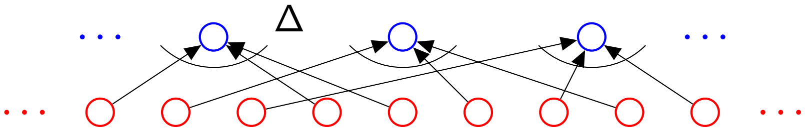
Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.

$\Delta = 3 \times 2 \times 1$ First rule again.
Total: $\frac{52!}{49!3!}$ Second Rule!

Choose k out of n.
Ordered set: $\frac{n!}{(n-k)!}$ Orderings of one hand?

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...

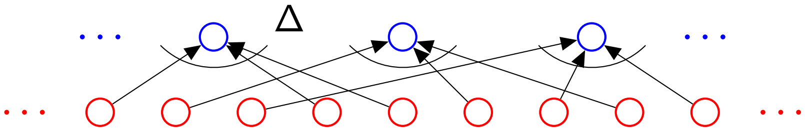


3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?
Hand: Q, K, A.
Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.
$\Delta = 3 \times 2 \times 1$ First rule again.
Total: $\frac{52!}{49!3!}$ Second Rule!

Choose k out of n.
Ordered set: $\frac{n!}{(n-k)!}$ Orderings of one hand? $k!$

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?

Hand: Q, K, A.

Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.

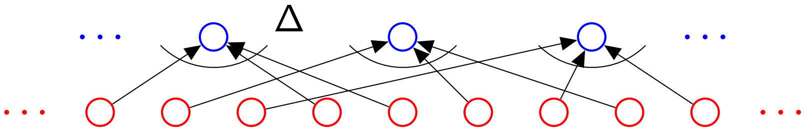
$\Delta = 3 \times 2 \times 1$ First rule again.
Total: $\frac{52!}{49!3!}$ Second Rule!

Choose k out of n.

Ordered set: $\frac{n!}{(n-k)!}$ Orderings of one hand? $k!$ (By first rule!)

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?

Hand: Q, K, A.

Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.

$\Delta = 3 \times 2 \times 1$ First rule again.
Total: $\frac{52!}{49!3!}$ Second Rule!

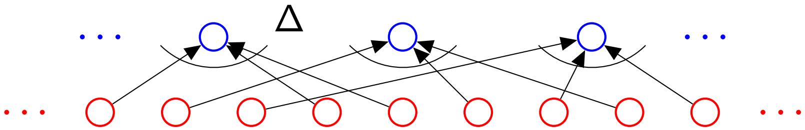
Choose k out of n.

Ordered set: $\frac{n!}{(n-k)!}$ Orderings of one hand? $k!$ (By first rule!)

$\implies$ Total: $\frac{n!}{(n-k)!k!}$

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?

Hand: Q, K, A.

Deals: $Q, K, A : Q, A, K : K, A, Q : K, A, Q : A, K, Q : A, Q, K$.

$\Delta = 3 \times 2 \times 1$ First rule again.
Total: $\frac{52!}{49!3!}$ Second Rule!

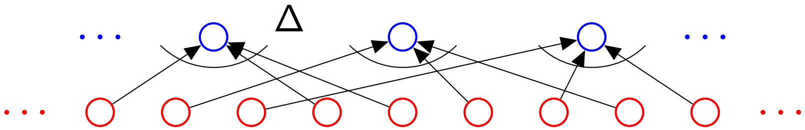
Choose k out of n.

Ordered set: $\frac{n!}{(n-k)!}$ Orderings of one hand? $k!$ (By first rule!)

$\Longrightarrow$ Total: $\frac{n!}{(n-k)!k!}$ Second rule.

Example: Visualize the proof..

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



3 card Poker deals: $52 \times 51 \times 50 = \frac{52!}{49!}$. First rule.
Poker hands: Δ?

Hand: Q, K, A.

Deals: $Q,K,A : Q,A,K : K,A,Q : K,A,Q : A,K,Q : A,Q,K$.

$\Delta = 3 \times 2 \times 1$ First rule again.
Total: $\frac{52!}{49!3!}$ Second Rule!

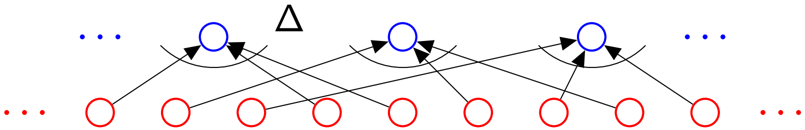
Choose k out of n.

Ordered set: $\frac{n!}{(n-k)!}$ Orderings of one hand? $k!$ (By first rule!)

$\Longrightarrow$ Total: $\frac{n!}{(n-k)!k!}$ Second rule.

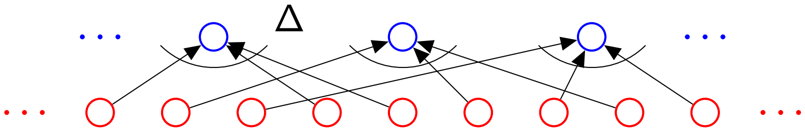
Example: Anagram

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



Example: Anagram

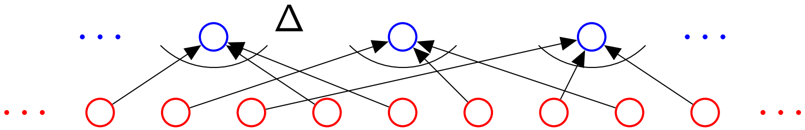
First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?

Example: Anagram

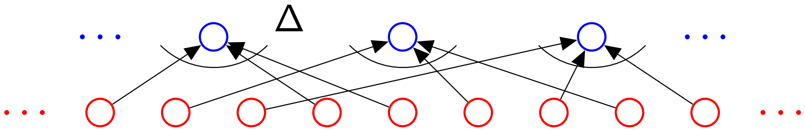
First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?
Ordered Set: 7!

Example: Anagram

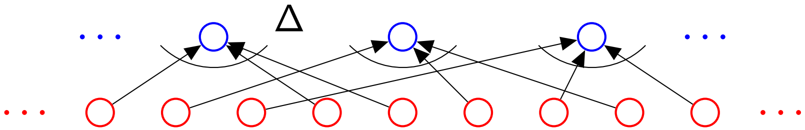
First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?
Ordered Set: 7! First rule.

Example: Anagram

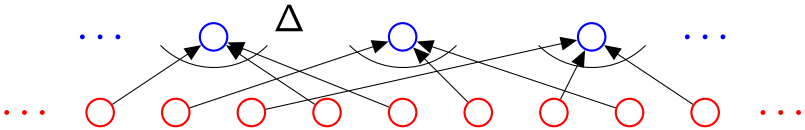
First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?
Ordered Set: 7! First rule.
A's are the same.

Example: Anagram

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?

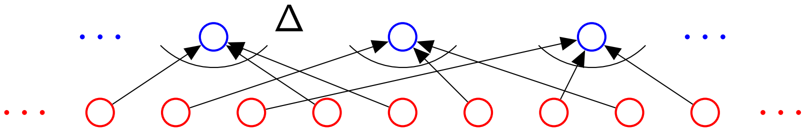
Ordered Set: 7! First rule.

A's are the same.

What is Δ?

Example: Anagram

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?

Ordered Set: 7! First rule.

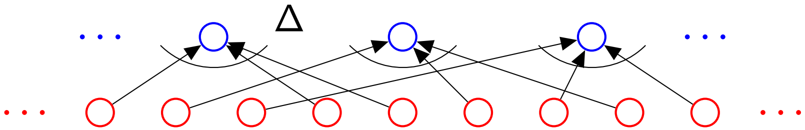
A's are the same.

What is Δ?

ANAGRAM

Example: Anagram

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?

Ordered Set: 7! First rule.

A's are the same.

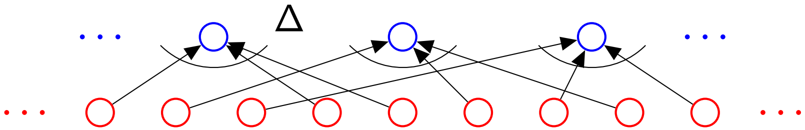
What is Δ?

ANAGRAM

$A_1NA_2GRA_3M$,

Example: Anagram

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?

Ordered Set: 7! First rule.

A's are the same.

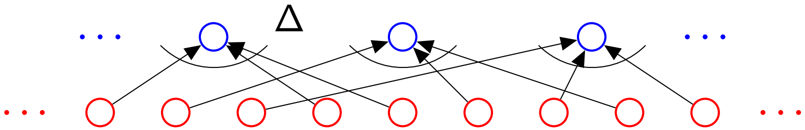
What is Δ?

ANAGRAM

$A_1NA_2GRA_3M$, $A_2NA_1GRA_3M$,

Example: Anagram

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?

Ordered Set: 7! First rule.

A's are the same.

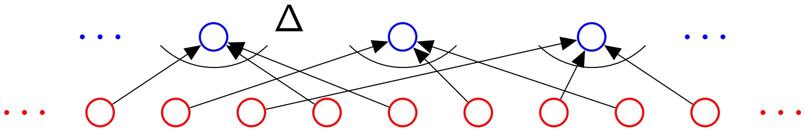
What is Δ?

ANAGRAM

$A_1NA_2GRA_3M$, $A_2NA_1GRA_3M$, ...

Example: Anagram

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?

Ordered Set: 7! First rule.

A's are the same.

What is Δ?

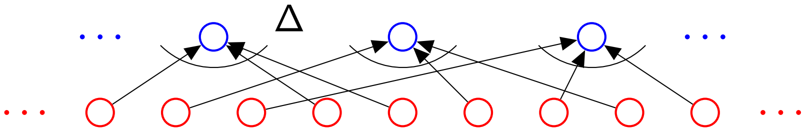
ANAGRAM

$A_1NA_2GRA_3M$, $A_2NA_1GRA_3M$, ...

$\Delta = 3 \times 2 \times 1$

Example: Anagram

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?

Ordered Set: 7! First rule.

A's are the same.

What is Δ?

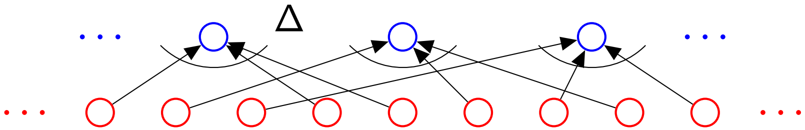
ANAGRAM

$A_1NA_2GRA_3M$, $A_2NA_1GRA_3M$, ...

$\Delta = 3 \times 2 \times 1 = 3!$

Example: Anagram

First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?

Ordered Set: 7! First rule.

A's are the same.

What is Δ?

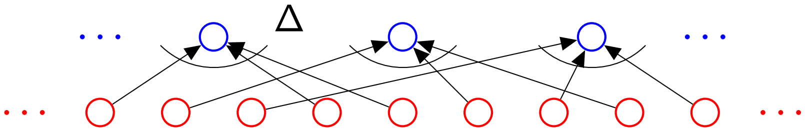
ANAGRAM

$A_1NA_2GRA_3M$, $A_2NA_1GRA_3M$, ...

$\Delta = 3 \times 2 \times 1 = 3!$ First rule!

Example: Anagram

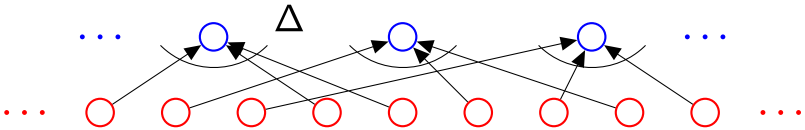
First rule: $n_1 \times n_2 \cdots \times n_3$**. Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?
Ordered Set: 7! First rule.
A's are the same.
What is Δ?
ANAGRAM
$A_1NA_2GRA_3M$, $A_2NA_1GRA_3M$, ...
$\Delta = 3 \times 2 \times 1 = 3!$ First rule!
$\implies \frac{7!}{3!}$

Example: Anagram

First rule: $n_1 \times n_2 \cdots \times n_3$. **Product Rule.**
Second rule: when order doesn't matter divide...



Orderings of ANAGRAM?
Ordered Set: 7! First rule.
A's are the same.
What is Δ?
ANAGRAM
$A_1NA_2GRA_3M$, $A_2NA_1GRA_3M$, ...
$\Delta = 3 \times 2 \times 1 = 3!$ First rule!
$\implies \frac{7!}{3!}$ Second rule!

Some Practice.

How many orderings of letters of CAT?

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1$

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\Longrightarrow$ $3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered,

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\Longrightarrow 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters.

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. 7!

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. 7!
total “extra counts” or orderings of three A’s?

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. 7!
total "extra counts" or orderings of three A's? 3!

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. 7!
total "extra counts" or orderings of three A's? 3!

Total orderings?

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. 7!
total "extra counts" or orderings of three A's? 3!

Total orderings? $\frac{7!}{3!}$

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. 7!
total "extra counts" or orderings of three A's? 3!

Total orderings? $\frac{7!}{3!}$

How many orderings of MISSISSIPPI?

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. 7!
total "extra counts" or orderings of three A's? 3!

Total orderings? $\frac{7!}{3!}$

How many orderings of MISSISSIPPI?

4 S's, 4 I's, 2 P's.

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. 7!
total "extra counts" or orderings of three A's? 3!

Total orderings? $\frac{7!}{3!}$

How many orderings of MISSISSIPPI?

4 S's, 4 I's, 2 P's.
11 letters total.

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. 7!
total “extra counts” or orderings of three A’s? 3!

Total orderings? $\frac{7!}{3!}$

How many orderings of MISSISSIPPI?

4 S’s, 4 I’s, 2 P’s.
11 letters total.
11! ordered objects.

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\implies 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. 7!
total "extra counts" or orderings of three A's? 3!

Total orderings? $\frac{7!}{3!}$

How many orderings of MISSISSIPPI?

4 S's, 4 I's, 2 P's.
11 letters total.
11! ordered objects.
$4! \times 4! \times 2!$ ordered objects per "unordered object"

Some Practice.

How many orderings of letters of CAT?

3 ways to choose first letter, 2 ways for second, 1 for last.

$\Longrightarrow 3 \times 2 \times 1 = 3!$ orderings

How many orderings of the letters in ANAGRAM?

Ordered, except for A!

total orderings of 7 letters. $7!$
total "extra counts" or orderings of three A's? $3!$

Total orderings? $\frac{7!}{3!}$

How many orderings of MISSISSIPPI?

4 S's, 4 I's, 2 P's.
11 letters total.
$11!$ ordered objects.
$4! \times 4! \times 2!$ ordered objects per "unordered object"

$\Longrightarrow \frac{11!}{4!4!2!}$.

More counting on Monday.