Introduction to Discrete Math

Felipe P. Vista IV

Reminder

- Everybody, make sure that your name in ZOOM is in the following format:
  – Ex: 202054321 Juan Dela Cruz

Not changing your name to this format

* you **will** be marked Absent * → absent?

Course Outline

- Mathematical Thinking
  - Convincing Arguments, Find Example, Recursion, Logic, Invariants
- Probability & Combinatronics
  - Counting, Probability, Random Variables
- Graph Theory
  - Graphs (cycles, classes, parameters)
- Number Theory & Cryptography
  - Arithmetic in modular form
  - Intro to Cryptography

Mathematical Thinking – Binomial Coefficients

COMBINATIONS

How Many?

- The main purpose is still to answer questions of the form “How many?”

How Many?

- The main purpose is still to answer questions of the form "How many?"
- We'll consider several non-trivial problems and we'll complement them with Python code

- Previously on Combinatorics
- Number of Games in a Tournament
- Combinations

Rule of Sum

Rule of Sum

If there are k objects of the first type and there are n objects of the second type, then there are $n + k$ objects of one of two types.

```
// OUTPUT
```

Rule of Sum

Rule of Sum

If there are k objects of the first type and there are n objects of the second type, then there are $n + k$ objects of one of two types.

```
A = [ 'Alice', 'Bob', 'Charlie']
B = [0 , 1 , 2 , 3]
print(A + B)
```

```
// OUTPUT
```

Rule of Sum

Rule of Sum

If there are k objects of the first type and there are n objects of the second type, then there are $n + k$ objects of one of two types.

```
A = [ 'Alice', 'Bob', 'Charlie']
B = [0 , 1 , 2 , 3]
print(A + B)
```

print (B + A)

```
// OUTPUT
['Alice','Bob','Charlie', 0 , 1 , 2 , 3]
```

Rule of Product

Rule of Product

If there are n objects of the first type & k objects of the second type, then there are $n \times k$ pairs of objects, the first of the first type & the second of the second type.

```
// OUTPUT
```

Rule of Product

Rule of Product

If there are n objects of the first type & k objects of the second type, then there are $n \times k$ pairs of objects, the first of the first type & the second of the second type.

```
from itertools import product
A = [’a’,’b’]
B = [1,2,3]
print(list(product(A,B)))
```

n × k A: 'a', 'b'
B: 1, 2, 3

```
// OUTPUT
```

Rule of Product

Rule of Product

If there are n objects of the first type & k objects of the second type, then there are $n \times k$ pairs of objects, the first of the first type & the second of the second type.

```
from itertools import product
A = ['a','b']
B = [1,2,3]
print(list(product(A,B)))
```

```
// OUTPUT
[('a',1),('a',2),('a',3),('b',1),('b',2),('b',3)]
```

Tuple

Tuple

The number of sequences of length k composed out of n symbols is n^k.

```
// OUTPUT
```

Tuple

Tuple

The number of sequences of length k composed out of n symbols is n^k.

```
from itertools import product
for p in product("ab", repeat=4):
    print("".join(p))
```

```
// OUTPUT
```

Tuple

Tuple

The number of sequences of length k composed out of n symbols is n^k.

```
from itertools import product
for p in product(”ab”, repeat=4):
    print(””.join(p))
```

```
// OUTPUT
aaaa
aaab
aaba
aabb
```

Tuple

Tuple

The number of sequences of length k composed out of n symbols is n^k.

```
from itertools import product
for p in product("ab", repeat=4):
    print("".join(p))
```

```
// OUTPUT
aaaa
aaab
aaba
aabb
```

```
abaa
abab
abba
abbb
```

Tuple

Tuple

The number of sequences of length k composed out of n symbols is n^k.

```
from itertools import product
for p in product(”ab”, repeat=4):
    print(””.join(p))
```

```
// OUTPUT
aaaa
aaab
aaba
aabb
```

```
abaa
abab
abba
abbb
```

```
baaa
baab
baba
babb
```

Tuple

Tuple

The number of sequences of length k composed out of n symbols is n^k.

```
from itertools import product
for p in product("ab", repeat=4):
    print("".join(p))
```

```
// OUTPUT
aaaa
aaab
aaba
aabb
```

```
abaa
abab
abba
abbb
```

```
baaa
baab
baba
babb
```

```
bbaa
bbab
bbba
bbbb
```

k-Permutations

k-Permutations

Number of sequences of length k with no repetitions composed of n symbols is $n(n-1)\cdots(n-k+1)=\frac{n!}{(n-k)!}$

```
// OUTPUT
```

k-Permutations

$\frac{5!}{(n-k)} = \frac{5!}{3!} = \frac{1\cdot2\cdot3\cdot4\cdot5}{1\cdot2\cdot3} = 20$

k-Permutations

Number of sequences of length k with no repetitions composed of n symbols is $n(n-1)\cdots(n-k+1) = \dfrac{n!}{(n-k)!}$

```
from itertools import permutations
for p in permutations("abcde",2):
    print("".join(p))
```

$n=5$ k

```
// OUTPUT
```

k-Permutations

k-Permutations

Number of sequences of length k with no repetitions composed of n symbols is $n(n-1)\cdots(n-k+1)=\frac{n!}{(n-k)!}$

```
from itertools import permutations
for p in permutations("abcde",2):
    print("".join(p))
```

```
// OUTPUT
ab
ac
ad
ae
```

k-Permutations

k-Permutations

Number of sequences of length k with no repetitions composed of n symbols is $n(n-1)\cdots(n-k+1) = \dfrac{n!}{(n-k)!}$

```
from itertools import permutations
for p in permutations(”abcde”,2):
    print(””.join(p))
```

```
// OUTPUT
ab
ac
ad
ae
```

```
ba
bc
bd
be
```

k-Permutations

k-Permutations

Number of sequences of length k with no repetitions composed of n symbols is $n(n-1)\cdots(n-k+1)=\frac{n!}{(n-k)!}$

```
from itertools import permutations
for p in permutations(”abcde”,2):
    print(””.join(p))
```

```
// OUTPUT
ab
ac
ad
ae
```

```
ba
bc
bd
be
```

```
ca
cb
cd
ce
```

k-Permutations

k-Permutations

Number of sequences of length k with no repetitions composed of n symbols is $n(n-1)\cdots(n-k+1)=\frac{n!}{(n-k)!}$

```
from itertools import permutations
for p in permutations("abcde",2):
    print("".join(p))
```

```
// OUTPUT
ab
ac
ad
ae
```

```
ba
bc
bd
be
```

```
ca
cb
cd
ce
```

```
da
db
dc
de
```

k-Permutations

k-Permutations

Number of sequences of length k with no repetitions composed of n symbols is $n(n-1)\cdots(n-k+1)=\dfrac{n!}{(n-k)!}$

```
from itertools import permutations
for p in permutations("abcde",2):
    print("".join(p))
```

```
// OUTPUT
ab
ac
ad
ae
```

```
ba
bc
bd
be
```

```
ca
cb
cd
ce
```

```
da
db
dc
de
```

```
ea
eb
ec
ed
```

Python Tip

Single Quote (') & Double Quote ("")

- Use depends on programmers preference
- To print double quotes, put inside single quote ' " '
- To print single quote, put inside double quotes " ' "
- Aside from previously given above, good rule to follow:
  - Double quotes for string representation
  - Single quote for regular expressions, dict keys, SQL
    - dict keys - Unordered data values in **key:value** pair format

- Previously on Combinatorics
- Number of Games in a Tournament
- Combinations

Tournament

Number of Games in a Tournament

Five teams played a tournament: each team played with each other. What was the total number of games played?

A E

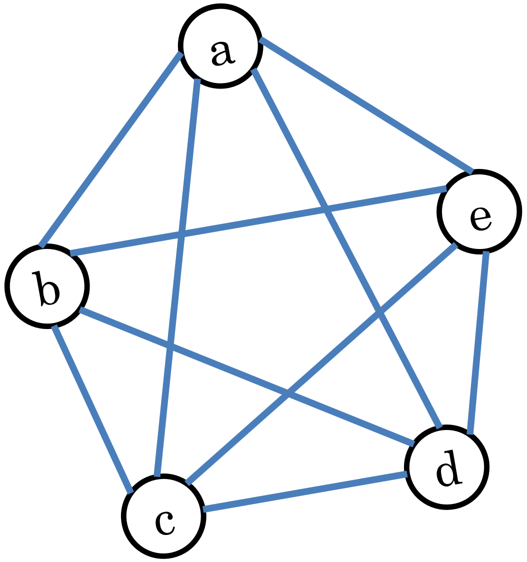
B D

C

Tournament

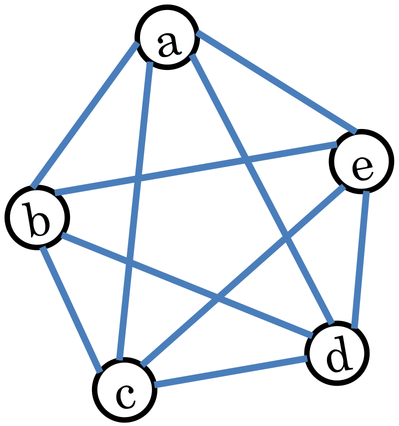
Number of Games in a Tournament

Five teams played a tournament: each team played with each other. What was the total number of games played?



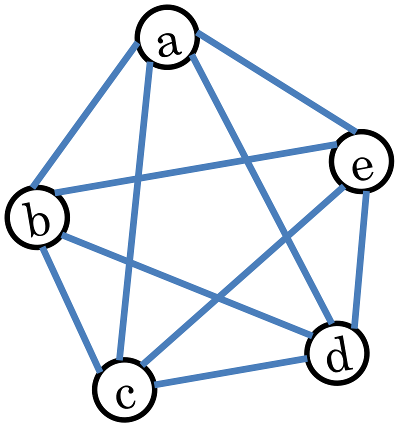
Tournament

Computing



Tournament

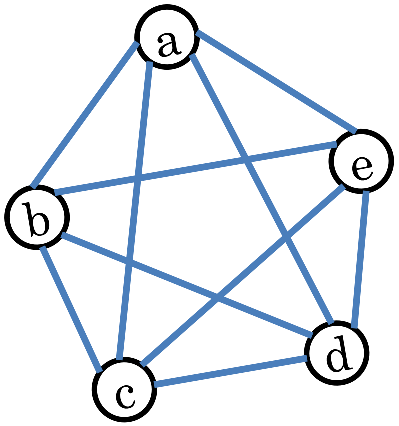
Computing



- Each team played four games

Tournament

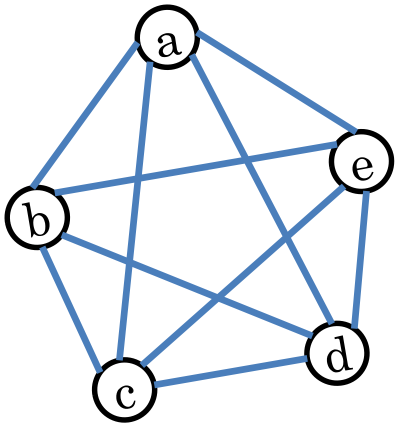
Computing



- Each team played four games
  - with each of the other four teams

Tournament

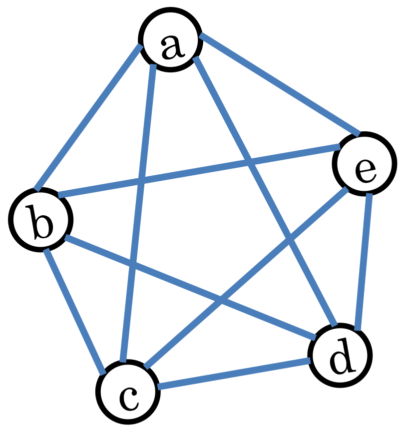
Computing



- Each team played four games
  - with each of the other four teams
- Hence, there were 5×4 games

Tournament

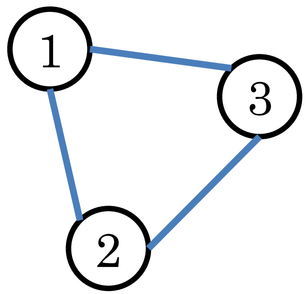
Computing



- Each team played four games
  - with each of the other four teams
- Hence, there were 5×4 games
- Do you see a flaw in this argument?

Tournament

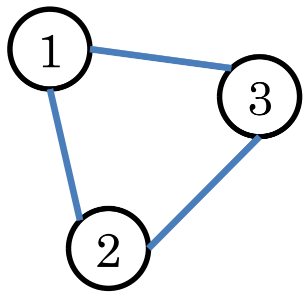
How about three teams?



Tournament

How about three teams?

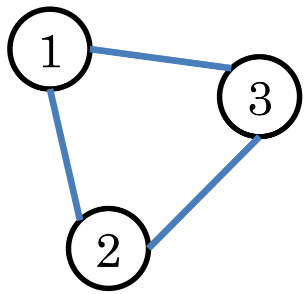
- For three teams, it gives us $3 \times 2 = 6$ games



Tournament

How about three teams?

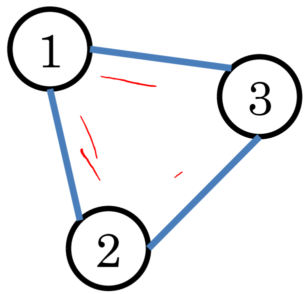
- For three teams, it gives us $3 \times 2 = 6$ games
- But in fact, there are 3 games



Tournament

How about three teams?

- For three teams, it gives us $3 \times 2 = 6$ games
- But in fact, there are 3 games
  - during each game, one of the teams takes a rest



Taking a Closer Look

- Our argument says that each of the five teams played with each of the four other teams

Taking a Closer Look

- Our argument says that each of the five teams played with each of the four other teams
- Denote the five teams a, b, c, d, & e; then consider the game matches as:

Taking a Closer Look

- Our argument says that each of the five teams played with each of the four other teams
- Denote the five teams a, b, c, d, & e; then consider the game matches as:

ab	ac	ad	ae
ba	bc	bd	be
ca	cb	cd	ce
da	db	dc	de
ea	eb	ec	ed

Taking a Closer Look

- Our argument says that each of the five teams played with each of the four other teams
- Denote the five teams a, b, c, d, & e; then consider the game matches as:

ab	**a**c	**a**d	**a**e
ba	bc	bd	be
ca	cb	cd	ce
da	db	dc	de
ea	eb	ec	ed

Taking a Closer Look

- Our argument says that each of the five teams played with each of the four other teams
- Denote the five teams a, b, c, d, & e; then consider the game matches as:

ab	ac	ad	ae
ba	**b**c	**b**d	**b**e
ca	cb	cd	ce
da	db	dc	de
ea	eb	ec	ed

Taking a Closer Look

- Our argument says that each of the five teams played with each of the four other teams
- Denote the five teams a, b, c, d, & e; then consider the game matches as:

ab	ac	ad	ae
ba	bc	bd	be
ca	**c**b	**c**d	**c**e
da	db	dc	de
ea	eb	ec	ed

Taking a Closer Look

- Our argument says that each of the five teams played with each of the four other teams
- Denote the five teams a, b, c, d, & e; then consider the game matches as:

ab	ac	ad	ae
ba	bc	bd	be
ca	cb	cd	ce
da	**d**b	**d**c	**d**e
ea	eb	ec	ed

Taking a Closer Look

- Our argument says that each of the five teams played with each of the four other teams
- Denote the five teams a, b, c, d, & e; then consider the game matches as:

ab	ac	ad	ae
ba	bc	bd	be
ca	cb	cd	ce
da	db	dc	de
ea	**e**b	**e**c	**e**d

Different Point of View

- Rearranging the games

```
ab  ac  ad  ae  bc  bd  be  cd  ce  de
ba  ca  da  ea  cb  db  eb  dc  ec  ed
```

Different Point of View

- Rearranging the games

```
ab ac ad ae bc bd be cd ce de
ba ca da ea cb db eb dc ec ed
```

- Do you notice something?

Different Point of View

- Rearranging the games

```
ab ac ad ae bc bd be cd ce de
ba ca da ea cb db eb dc ec ed
```

- Do you notice something?
- Each game is counted twice!

Different Point of View

- Rearranging the games

  ab ac ad ae bc bd be cd ce de

  ba ca da ea cb db eb dc ec ed

- Do you notice something?
- Each game is counted twice!

Different Point of View

- Rearranging the games

  ab ac ad ae bc bd be cd ce de

  ba ca da ea cb db eb dc ec ed

- Do you notice something?
- Each game is counted twice!
- Thus, the actual total number of games will be

Different Point of View

- Rearranging the games

  ab ac ad ae bc bd be cd ce de
  ba ca da ea cb db eb dc ec ed

- Do you notice something?
- Each game is counted twice!
- Thus, the actual total number of games will be
  - *(5 x 4) / 2 = 10*

Important Message

- When counting

Important Message

- When counting
  - make sure that each object is counted only once

Important Message

- When counting
  - make sure that each object is counted only once
- If each object is counted k times

Important Message

- When counting
  - make sure that each object is counted once
- If each object is counted k times
  - divide the resulting count by k

Formally

Theorem

The number of games in a tournament with n teams is $n(n-1)/2$ wherein each pair of teams play against each other exactly once.

Formally

Proof

- There are n choices of the first team in a game and correspondingly $(n-1)$ choices of the second team

Formally

Proof

- There are n choices of the first team in a game and correspondingly $(n - 1)$ choices of the second team
- Each game is counted twice:

Formally

Proof

- There are n choices of the first team in a game and correspondingly $(n - 1)$ choices of the second team
- Each game is counted twice:
  - the game between teams i and j is counted as ij and as ji

Formally

Proof

- There are n choices of the first team in a game and correspondingly $(n-1)$ choices of the second team
- Each game is counted twice:
  - the game between teams i and j is counted as ij and as ji
- Thus, the total number of games is $n(n-1)/2$

if $n = 3$; $3(3-1)/2 \Rightarrow 3\cancel{(2)}/\cancel{2} = 3$

Formally

Another Proof

- Let’s count the number of games recursively

Formally

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

Another Proof

- Let's count the number of games recursively
- Denote by $\boldsymbol{T(n)}$ the number of games in a tournament with n teams

Formally

Another Proof

- Let's count the number of games recursively
- Denote by $\boldsymbol{T(n)}$ the number of games in a tournament with n teams
- There are two types of games:

Formally

Another Proof

- Let's count the number of games recursively
- Denote by $\boldsymbol{T(n)}$ the number of games in a tournament with n teams
- There are two types of games:
  - games that involve the first team: $\boldsymbol{(n-1)}$

Formally

Another Proof

- Let's count the number of games recursively
- Denote by $T(n)$ the total number of games in a tournament with n teams
- There are two types of games:
  - games that involve the first team: $(n - 1)$ a
  - games that doesn't involve them: $T(n - 1)$ a x

Formally

Another Proof

- Let's count the number of games recursively
- Denote by $T(n)$ the number of games in a tournament with n teams
- There are two types of games:
  - games that involve the first team: $(n-1)$
  - games that doesn't involve them: $T(n-1)$
- Hence, $T(n) = (n-1) + T(n-1)$

\# "a" played "a" did not play

Unwinding the Recurrence Relation

$$T(n) = (n-1) + T(n-1)$$

Unwinding the Recurrence Relation

Total # Games

$$T(n) = \underbrace{(n-1)}_{\text{all "A" played}} + \underbrace{T(n-1)}_{\text{all "A" did not play}};but\ T(n-1) = \underbrace{(n-2)}_{\text{all games w/ "B"}} + \underbrace{T(n-2)}_{\text{all games w/out "B"}}$$

all games "A" not played

Unwinding the Recurrence Relation

$$T(n) = (n-1) + T(n-1); but \quad T(n-1) = (n-2) + T(n-2)$$

$$\therefore$$

$$T(n) = (n-1) + T(n-1)$$

$$= (n-1) + (n-2) + T(n-2)$$

Unwinding the Recurrence Relation

$$T(n) = (n-1) + T(n-1)$$

$$= (n-1) + (n-2) + T(n-2); but \;\; T(n-2) = (n-3) + T(n-3)$$

games "A" played

games "B" played

games "B" did not play

games "C" played

games "C" did not play

Unwinding the Recurrence Relation

$$T(n)=(n-1)+T(n-1)$$
$$=(n-1)+(n-2)+T(n-2);but\ T(n-2)=(n-3)+T(n-3)$$
$$\therefore$$
$$T(n)=(n-1)+T(n-1)$$
$$=(n-1)+(n-2)+T(n-2)$$
$$=(n-1)+(n-2)+(n-3)+T(n-3)$$

Unwinding the Recurrence Relation

$$
\begin{aligned}
T(n) &= (n-1) + T(n-1) \\
&= (n-1) + (n-2) + T(n-2); but \ T(n-2) = (n-3) + T(n-3) \\
&\therefore \\
T(n) &= (n-1) + T(n-1) \\
&= (n-1) + (n-2) + T(n-2) \\
&= (n-1) + (n-2) + (n-3) + T(n-3) \\
&= \cdots \\
&= (n-1) + (n-2) + \cdots + 2 + 1 + 0
\end{aligned}
$$

"A" placed "B" placed "C" placed "C" "A" "D" dist

2 1 0

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

Arithmetic Series

- Compute the sum of the series with itself reversed:

$$\begin{array}{cccccc} & (n-\cancel{1}) & (n-\cancel{2}) & \cdots & \cancel{2} & \cancel{1} \\ + & \cancel{1} & +\cancel{2} & \cdots & (n-\cancel{2}) & (n-\cancel{1}) \\ \hline & n & n & n & n & n \end{array}$$

Arithmetic Series

- Compute the sum of the series with itself reversed:

$$
\begin{array}{ccccc}
(n-1) & (n-2) & \cdots & 2 & 1 \\
1 & 2 & \cdots & (n-2) & (n-1) \\
\hline
n & n & n & n & n
\end{array}
$$

- Hence, $\boldsymbol{T(n) = n(n-1)/2}$

Code (Python)

```
from itertools import combinations

for c in combinations(”abcdefgh”,2):
print(””.join(c))
```

```
// OUTPUT
```

Code (Python)

```
from itertools import combinations
for c in combinations(”abcdefgh”,2):
print(””.join(c))
```

from import permutations → import itertools as it

→ it.combinations(..
it.permutations( )

```
// OUTPUT
ab
ac
ad
ae
af
ag
ah
bc
bd
be
bf
bg
bh
cd
ce
cf
cg
ch
de
df
dg
dh
ef
eg
eh
fg
fh
gh
```

Code (Python)

Recursion

```
def T(n):
    if n <= 1:
        return 0
    return(n - 1) + T ( n - 1)
print(T(8))
```

```
// OUTPUT
```

Code (Python)

Recursion

```
def T(n):
    if n <= 1:
        return 0
    return(n - 1) + T ( n - 1)
print(T(8))
```

```
// OUTPUT
28
```

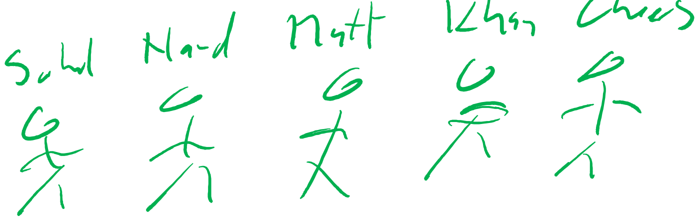
- Previously on Combinatorics
- Number of Games in a Tournament
- Combinations

Counting Subsets

Road Trip!

You are organizing a road trip. You have five friends, but there are only three vacant places in you car. What is the number of ways of taking three of your five friends to the journey?

set A =



set B = subset(A, 3)

SMM . MSK
SMK MSC
SNC

Counting Subsets

Road Trip!

You are organizing a road trip. You have five friends, but there are only three vacant places in you car. What is the number of ways of taking three of your five friends to the journey?

Subsets

What is the number of ways of choosing 3 elements out of a set of size 5?

Counting

- There are five ways to choose first friend, four ways the second friend, and three for third friend to bring

S M N K C

1st seat

2nd seat

3rd seat

$n = 5$ choices

$n-1 = 4$ choices

$n-2 = 3$ choices

Counting

- There are five ways to choose first friend, four ways the second friend, and three for third friend to bring
  - In total $5 \times 4 \times 3 = 60$ choices

Counting

- There are five ways to choose first friend, four ways the second friend, and three for third friend to bring
  - In total $5 \times 4 \times 3 = 60$ choices
- But each group of three friends is counted $3! = 6$ times:

Counting

- There are five ways to choose first friend, four ways the second friend, and three for third friend to bring
  - In total $5 \times 4 \times 3 = 60$ choices
- But each group of three friends is counted $3! = 6$ times:
  - a group $\{a, b, c\}$ is counted as $abc, acb, bac, bca, cab, cba$

Counting

- There are five ways to choose first friend, four ways the second friend, and three for third friend to bring
  - In total $5 \times 4 \times 3 = 60$ choices
- But each group of three friends is counted
  $3! = 6$ times:
  - a group $\{a, b, c\}$ is counted as abc, acb, bac, bca, cab, cba
- Thus, the answer is $(5 \times 4 \times 3)/3! = 10$

Code (Python)

```
import itertools as it

for c in it.combinations(”abcde”,3):
   print(””.join(c))
```

```
// OUTPUT
```

Code (Python)

```
import itertools as it

for c in it.combinations(”abcde”,3):
   print(””.join(c))
```

```
// OUTPUT
bca
abd
abe
acd
ace
```

```
ade
bcd
bce
bde
cde
```

Combinations

Definition

For a set S, its *k-combination* is a subset of S of size k.

Combinations

Definition

For a set S, its *k-combination* is a subset of S of size k.

Subsets

The number of *k-combinations* of an n element set is denoted by (nk).

Combinations

Definition

For a set S, its *k-combination* is a subset of S of size k.

Subsets

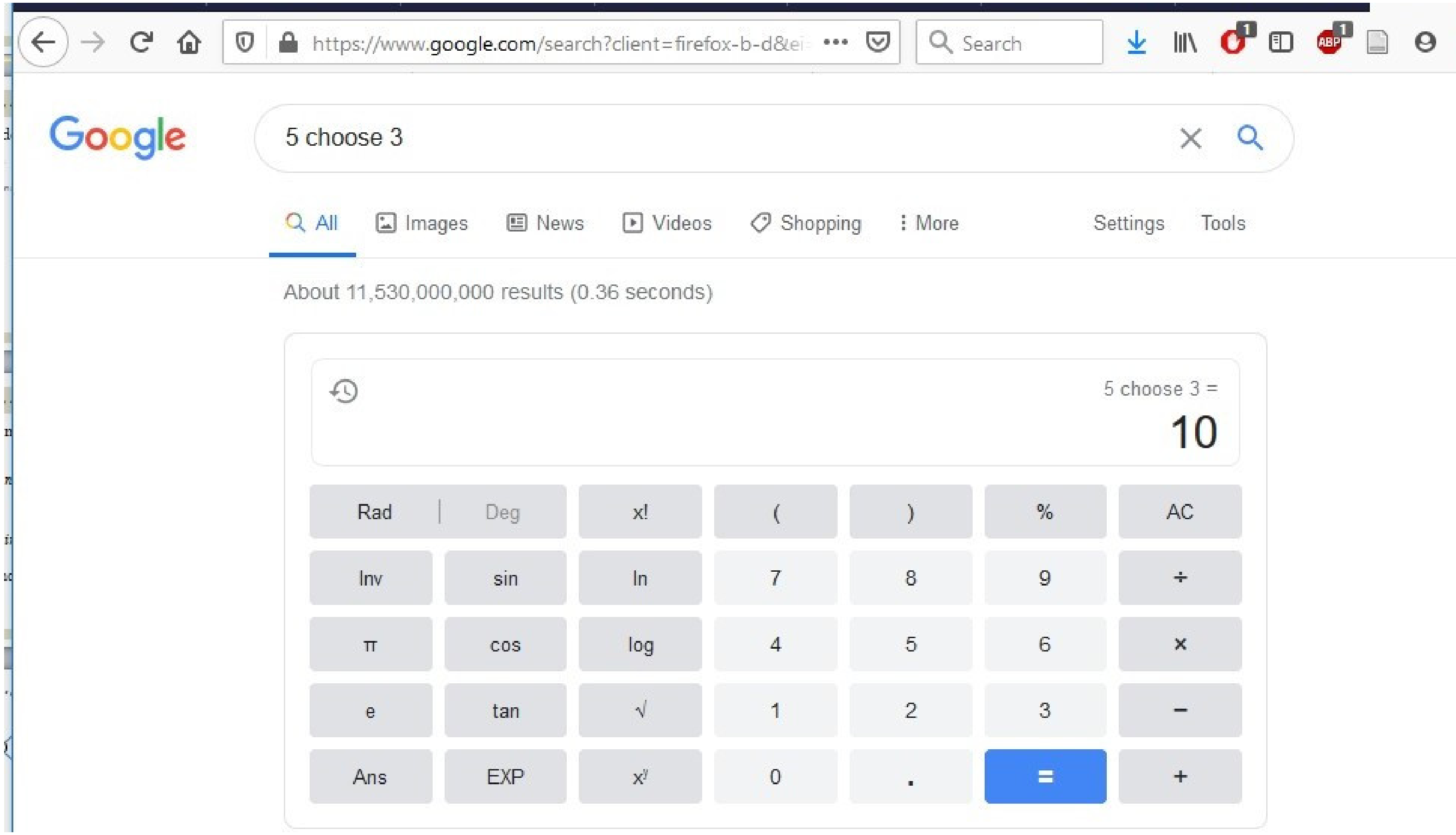
The number of *k-combinations* of an n element set is denoted by *(nk)*.

- Pronounced as: "***n* choose *k***".

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

if $n=5, k=3 \Rightarrow \frac{5!}{3!(5-3)!} = \frac{5!}{3!(2!)} \Rightarrow \frac{\cancel{1\cdot 2\cdot 3}\cdot 4\cdot 5}{\cancel{(1\cdot 2\cdot 3)}(1\cdot 2)} = \frac{20}{2} = 10$

“*n choose k*” in GOOGLE



3-Combinations & 3-Permutations

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- Recall total number of *3* ***k-permutations*** from *5* symbols

3-Combinations & 3-Permutations

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- Recall total number of *3* ***k-permutations*** from *5* symbols
- $5 \times 4 \times 3 = 60$

3-Combinations & 3-Permutations

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- Recall total number of *3* ***k-permutations*** from *5* symbols
- *5 x 4 x 3 = 60*
- All possible *3 k-permutations* given above

3-Combinations & 3-Permutations

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

3-Combinations & 3-Permutations

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- row shows all possible ***k-combinations;*** subset 3 out of 5 symbols

3-Combinations & 3-Permutations

$\binom{5}{3}$

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- row shows all possible ***k-combinations***; subset 3 out of 5 symbols
  - size: $\binom{5}{3}$ (5 choose 3) $\rightarrow$ 10

3-Combinations & 3-Permutations

$\binom{5}{3}$

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- row shows all possible ***k-combinations***; subset 3 out of 5 symbols
  - size: $\binom{5}{3}$ (5 choose 3) → 10

Recall: $n \text{ choose } k = \dfrac{n!}{k!(n-k)!}$

3-Combinations & 3-Permutations

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- column shows all possible ***k-permutations;*** subset 3 out of 5 symbols

3-Combinations & 3-Permutations

	abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
	acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
3!	bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
	bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
	cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
	cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- column shows all possible ***k-permutations;*** subset 3 out of 5 symbols
  - Size: 3! (3 factorial) → 6

3-Combinations & 3-Permutations

$\binom{5}{3}$

3!

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- row shows all possible ***k-combinations;*** subset 3 out of 5 symbols
  - size: $\binom{5}{3}$ (5 choose 3) → 10
- column shows all possible ***k-permutations;*** subset 3 out of 5 symbols
  - Size: 3! (3 factorial) → 6

3-Combinations & 3-Permutations

$\binom{5}{3}$

3!

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- row shows all possible ***k-combinations;*** subset 3 out of 5 symbols
  - size: $\binom{5}{3}$ (5 choose 3) → 10
- column shows all possible ***k-permutations;*** subset 3 out of 5 symbols
  - Size: 3! (3 factorial) → 6

$$3!\binom{5}{3} = \frac{5!}{(5-3)!}$$

3-Combinations & 3-Permutations

$\binom{5}{3}$

3!

abc	abd	abe	acd	ace	ade	bcd	bce	bde	cde
acb	adb	aeb	adc	aec	aed	bdc	bec	bed	ced
bac	bad	bae	cad	cae	dae	cbd	cbe	dbe	dce
bca	bda	bea	cda	cea	dea	cdb	ceb	deb	dec
cba	dba	eba	dca	eca	eda	dcb	ecb	edb	edc
cab	dab	eab	dac	eac	ead	dbc	ebc	ebd	ecd

- row shows all possible ***k-combinations;*** subset 3 out of 5 symbols
  - size: $\binom{5}{3}$ (5 choose 3) $\rightarrow$ 10
- column shows all possible ***k-permutations;*** subset 3 out of 5 symbols
  - Size: 3! (3 factorial) $\rightarrow$ 6

$$3!\binom{5}{3} = \frac{5!}{(5-3)!} \Rightarrow \frac{5!}{2!} = \frac{5\times4\times3\times\cancel{2!}}{\cancel{2!}} = 5\times4\times3 = 60$$

Number of Combinations

Theorem

n choose k-combinations formula

$$\binom{n}{k}_{combinations} = \frac{n!}{k!(n-k)!}$$

Number of Combinations

Proof

- There are:
  - n possible choice for the first element,
  - $(n - 1)$ possible choices for the second element, : : : ,
  - $(n - k + 1)$ possible choices for the k-th element

$n - \cancel{1+1} \Rightarrow n$; if $n=2 \Rightarrow n-2+1 = n-1$

Number of Combinations

Proof

- There are:
  - n possible choice for the first element,
  - $(n-1)$ possible choices for the second element, : : : ,
  - $(n-k+1)$ possible choices for the k-th element
- This gives $n(n-1)\cdots(n-k+1)=\dfrac{n!}{(n-k)!}$

Number of Combinations

Proof

- There are:
  - n possible choice for the first element,
  - $(n-1)$ possible choices for the second element, : : : ,
  - $(n-k+1)$ possible choices for the k-th element
- This gives $n(n-1)\cdots(n-k+1)=\dfrac{n!}{(n-k)!}$
- But this is the number of ***k-permutations*** rather than the number of ***k-combinations***

Number of Combinations

Proof

- There are:
  - n possible choice for the first element,
  - $(n - 1)$ possible choices for the second element, : : : ,
  - $(n - k + 1)$ possible choices for the k-*th* element
- This gives $n(n-1)\cdots(n-k+1)=\dfrac{n!}{(n-k)!}$
- But this is the number of ***k-permutations*** rather than the number of ***k-combinations***
- Each subset is counted $k!$ times

Number of Combinations

Proof

- There are:
  - n possible choice for the first element,
  - $(n-1)$ possible choices for the second element, : : : ,
  - $(n-k+1)$ possible choices for the k-th element
- This gives $n(n-1)\cdots(n-k+1)=\dfrac{n!}{(n-k)!}$
- But this is the number of ***k-permutations*** rather than the number of ***k-combinations***
- Each subset is counted $k!$ times
- This finally gives $\dfrac{n!}{k!(n-k)!}$

Thank you.