

We will be writing a
lot of code today.
Have your IDE ready!



Week 9 Tutorial

COMP10001 – Foundations of Computing

Semester 1, 2025

Clement Chau

- List Comprehensions
- Iterators and Itertools

Important Project 1 Announcement:



Huey Yee Chan AUTHOR | INSTRUCTOR

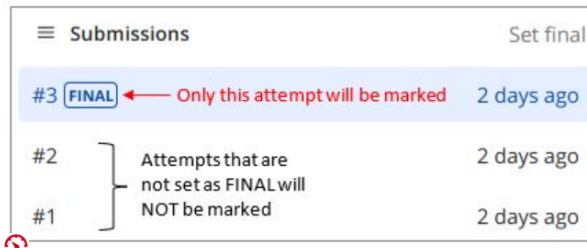
Created May 5 0:32 | Posted May 5 0:32



Project 1 FINAL submission notice

Dear students,

This is a notification regarding submissions for Project 1. Please be reminded that **ONLY** attempts set as **FINAL** for each task will be marked, see screenshot in Ed below.



By default, when submissions closed on the deadline, the most recent attempt will be set as the FINAL submission. For students who have multiple attempts (submissions), please check that the one set as FINAL is the one that you wish to be marked.

If you want us to mark a different submission from the one that is currently set as FINAL, please submit a request to change the FINAL submission to a different attempt via this form,

<https://forms.office.com/r/C3QzfZVy6C>

You will need to specify the Task that you wish for a submission to be amended and the Submission# that you wish to be set as FINAL. Example Task 2 Submission #2 from the screenshot above.

The request form will close on **Tuesday, 6 May 2025, at 10.00am**.

Regards,

The COMP10001 teaching team

Revision: List Comprehensions


```
cashier_3 = []  
for item in cart:  
    if item % 2 == 0:  
        cashier_3.append(item)
```

Non-list comprehension

```
cashier_3 = [item for item in cart if item % 2 == 0]
```

List comprehension

Revision: (Types of) List Comprehensions

sem1-2025 > week-9 >  list_comprehension.py > ...

```
1  # List Comprehension
2  l = [i for i in range(10)]
3
4  # List Comprehension with If Condition
5  l = [i for i in range(10) if i % 2 == 0]
6
7  # List Comprehension with If/Else Condition
8  l = [i if i % 2 == 0 else i * i for i in range(10)]
9
10 # Nested Comprehension
11 l = [(i, j) for i in range(10) for j in range(10)]
```

Exercise 1 / 9

Evaluate the following list comprehensions. For each one, also write some python code to generate the same list without using a comprehension.

(a) `[(name, 0) for name in ("evelyn", "alex", "sam")]`

(b) `[i**2 for i in range(5) if i % 2 == 1]`

(c) `"".join([letter.upper() for letter in "python"])`

(d) `[(row, col) for row in range(3, 5) for col in range(2)]`

A:

```
[('evelyn', 0), ('alex', 0), ('sam', 0)]
```

(a)

```
my_list = []
for name in ("evelyn", "alex", "sam"):
    my_list.append((name, 0))
```

A:

```
[1, 9]
```

(b)

```
my_list = []
for i in range(5):
    if i % 2 == 1:
        my_list.append(i**2)
```

Answer:

Exercise 1 / 9

Evaluate the following list comprehensions. For each one, also write some python code to generate the same list without using a comprehension.

(a) `[(name, 0) for name in ("evelyn", "alex", "sam")]`

(b) `[i**2 for i in range(5) if i % 2 == 1]`

(c) `"".join([letter.upper() for letter in "python"])`

(d) `[(row, col) for row in range(3, 5) for col in range(2)]`

A:

```
'PYTHON'
```

(c)

```
my_list = []
for letter in "python":
    my_list.append(letter.upper())
my_str = "".join(my_list)
```

We could simplify this code to just `my_str = "python".upper()` if we don't require the list!

A:

```
[(3, 0), (3, 1), (4, 0), (4, 1)]
```

(d)

```
my_list = []
for row in range(3, 5):
    for col in range(2):
        my_list.append((row, col))
```

Answer:

Exercise 2 / 9

2. What happens if we use curly brackets instead of square brackets around a “list” comprehension? What happens if we use parentheses?

```
13  # Dictionary Comprehension
14  d = {i: i * i for i in range(10)}
15
16  # Set Comprehension
17  s = {i for i in range(10) if i % 2 == 0}
18
19  # Generator Expression (Not Examinable)
20  g = (i for i in range(10) if i % 2 == 0)
21
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
{0, 2, 4, 6, 8}
<generator object <genexpr> at 0x000002C6921235E0>
```

Exercise 3 / 9

3. For a list such as `words = ['pencil', 'highlighter', 'paper-clip', 'ruler', 'pen']`, write a comprehension that gives the following:

- (a) A list containing only the words that start with 'p'
- (b) A dictionary mapping each word to their length
- (c) A set of every character used in words

A:

(a) `[word for word in words if word.startswith('p')]`

output: `['pencil', 'paper-clip', 'pen']`

A:

(b) `{word: len(word) for word in words}`

output: `{ 'pencil':6, 'highlighter':11, 'paper-clip':10, 'ruler':5, 'pen':3}`


A:

(c) `{letter for word in words for letter in word}`

output: `{ '-', 'a', 'c', 'e', 'g', 'h', 'i', 'l', 'n', 'p', 'r', 't', 'u'}`


Answer:

Revision: Iterators

```
▶ Run PYTHON 
```

```
1 list1 = [1, 5]
2 my_iterator = iter(list1)
3 print(next(my_iterator))
4 print(next(my_iterator))
5
6 # This will generate an error because we have reached the end.
7 print(next(my_iterator))
```

1
5
Traceback (most recent call last):
 File "/home/main.py", line 7, in <module>
 print(next(my_iterator))
 ~~~~^  
**StopIteration**

 Program exited with code 1

# Revision: Iterables

- **list**
- **str**
- **tuple**
- **set**
- **dict**
- file objects (We'll talk more about this next week!)

# Revision: Iterators vs Sequences

## Sequences:

- Have *random access* (you can access any element in the sequence, as many times as you like)
- No position tracking within the sequence
- You can use `len()` to calculate the length
- Must be finite
- You can traverse it many times

## Iterators:

- No *random access*
- Remembers where you are up to
- Cannot use `len()`
- Can be infinite
- You can only traverse it once, forwards.

# Exercise 4 / 9

Convert these iterable objects into iterators and extract two elements into `first` and `second` variables:

(a) `iterable = "ABCDEFGH"`

(b) `iterable = {(0, 0), (0, 1), (1, 0), (1, 1)}`

(a) A:

```
iterable = "ABCDEFGH"
iterator = iter(iterable)
first = next(iterator) # 'A'
second = next(iterator) # 'B'
```

*In this case, `iterable` is a `str`. Calling `iter()` on a string creates an iterator that returns one character at a time, starting from index 0, and `next()` retrieves the next character from the iterator.*

**Answer:**

(b) A:

```
iterable = {(0, 0), (0, 1), (1, 0), (1, 1)}
iterator = iter(iterable)
first = next(iterator) #e.g., (0, 1)
second = next(iterator) #e.g., (1, 0)
```

*In this case, `iterable` is a `set` of tuples. Since sets are unordered, the iteration order is not guaranteed. Calling `iter()` on a set creates an iterator that returns the elements one by one, and `next()` retrieves the next tuple from the iterator.*

# Revision: Itertools (Cycle)

▶ Run

PYTHON



```
1 from itertools import cycle
2 COUNT = 4
3 my_iterator = cycle("ABC")
4 for _i in range(COUNT):
5     print(next(my_iterator))
```

A  
B  
C  
A

✓ Program exited with code 0

# Exercise 6 / 9

6. What output does the following code print? Try changing the `while` loop to get the same output.

```
import itertools
beatboxer = itertools.cycle(['boots', 'and', 'cats', 'and'])

for count in range(9):
    print(next(beatboxer))
```

**A:** *This code will print two iterations of boots and cats and, which will end with boots:*

```
boots
and
cats
and
...
boots
```

## Answer:

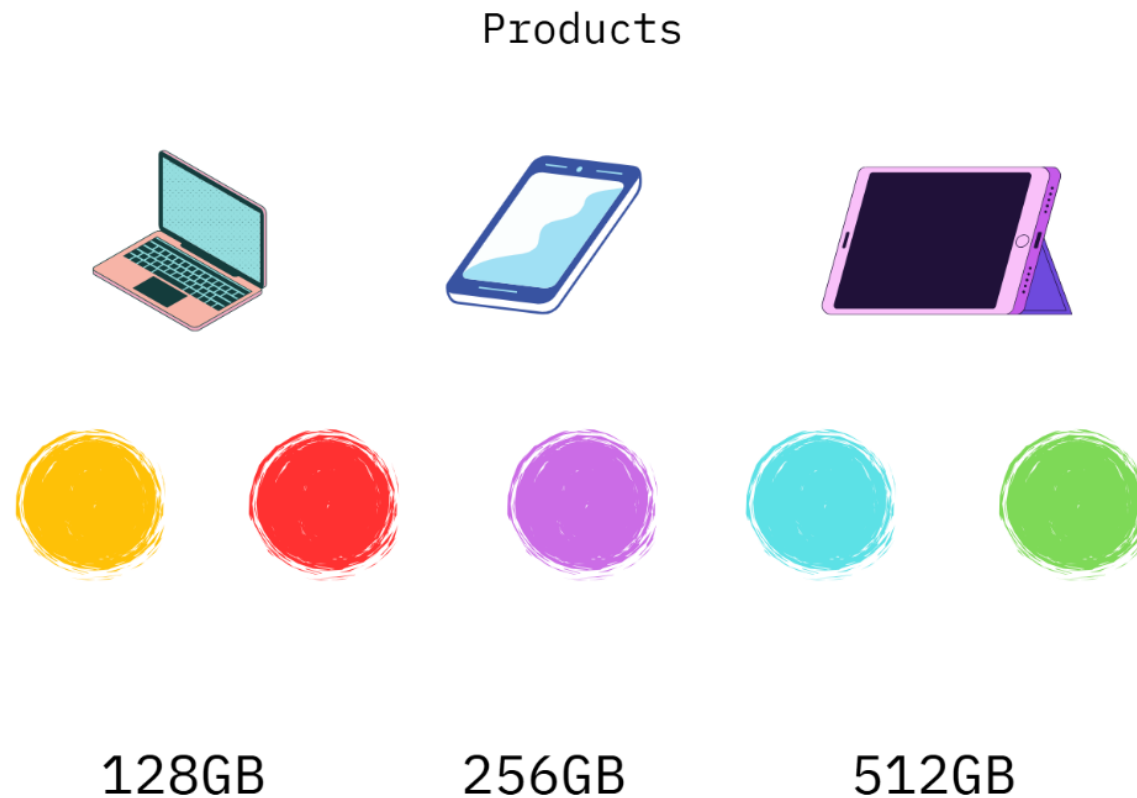
```
import itertools
beatboxer = itertools.cycle(['boots', 'and', 'cats', 'and'])

COUNT = 9
while COUNT:
    print(next(beatboxer))
    COUNT -= 1
```

*Also, try changing the `while True:` (an infinite loop) to see this cycle print infinitely!*

# Revision: Itertools (Product)

I have these options for devices, colors, and storage size. How many apple products can I come up with?



# Revision: Itertools (Product)

```
1 import itertools
2
3 all_products = itertools.product(
4     ['laptop', 'phone', 'tablet'],
5     ['Y', 'R', 'P', 'B', 'G'],
6     [128, 256, 512]
7 )
8
9 print(list(all_products))
```

```
[('laptop', 'Y', 128), ('laptop', 'Y', 256), ('laptop', 'Y', 512), ('laptop', 'R', 128), ('laptop', 'R', 256), ('laptop', 'R', 512), ('laptop', 'P', 128), ('laptop', 'P', 256), ('laptop', 'P', 512), ('laptop', 'B', 128), ('laptop', 'B', 256), ('laptop', 'B', 512), ('laptop', 'G', 128), ('laptop', 'G', 256), ('laptop', 'G', 512), ('phone', 'Y', 128), ('phone', 'Y', 256), ('phone', 'Y', 512), ('phone', 'R', 128), ('phone', 'R', 256), ('phone', 'R', 512), ('phone', 'P', 128), ('phone', 'P', 256), ('phone', 'P', 512), ('phone', 'B', 128), ('phone', 'B', 256), ('phone', 'B', 512), ('phone', 'G', 128), ('phone', 'G', 256), ('phone', 'G', 512), ('tablet', 'Y', 128), ('tablet', 'Y', 256), ('tablet', 'Y', 512), ('tablet', 'R', 128), ('tablet', 'R', 256), ('tablet', 'R', 512), ('tablet', 'P', 128), ('tablet', 'P', 256), ('tablet', 'P', 512), ('tablet', 'B', 128), ('tablet', 'B', 256), ('tablet', 'B', 512), ('tablet', 'G', 128), ('tablet', 'G', 256), ('tablet', 'G', 512)]
```



# Exercise 7 / 9

7. A comedy series has episode names in an <animal> in <place> format, for example, “Elephants in Melbourne”. Using a single loop, write some code to print out every possible episode name, given:

```
animals = ['cats', 'dogs', 'hamsters', 'elephants']  
places = ['Melbourne', 'space', 'the supermarket']
```

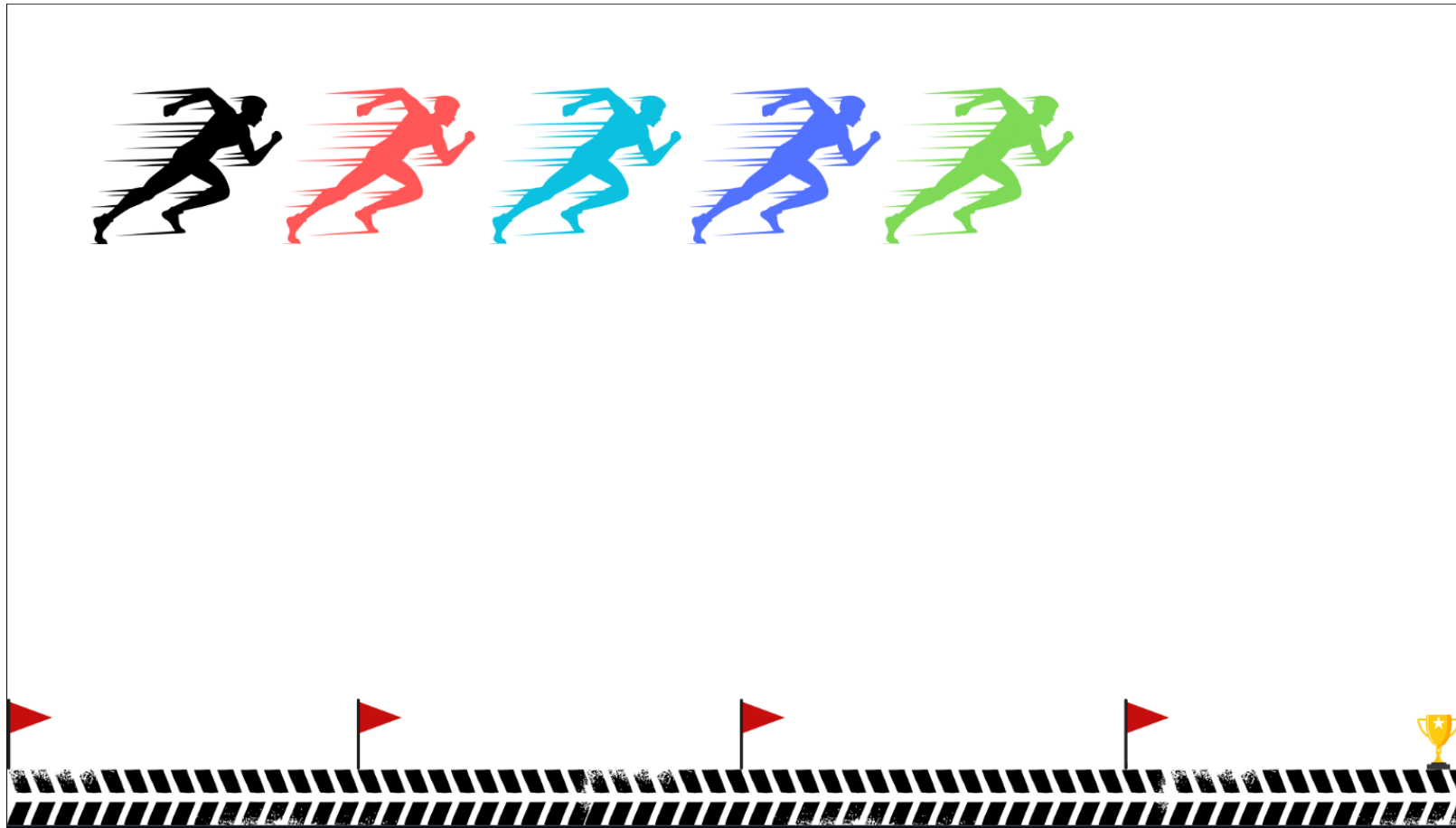
A:

**Answer:**

```
import itertools  
  
animals = ['cats', 'dogs', 'hamsters', 'elephants']  
places = ['Melbourne', 'space', 'the supermarket']  
  
for animal, place in itertools.product(animals, places):  
    print(f"{animal.title()} in {place}")
```

# Revision: Itertools (Permutations)

I want to know all possible orderings for a 4-man relay sprint from 5 candidates



Credit: Daksh Agrawal

# Revision: Itertools (Permutations)

```
1 import itertools
2
3 p_teams = itertools.permutations(
4     iterable: ["A", "B", "C", "D", "E"],
5     r: 4
6 )
7
8 print(list(p_teams))
```

```
[('A', 'B', 'C', 'D'), ('A', 'B', 'C', 'E'), ('A', 'B', 'D', 'C'), ('A', 'B', 'D', 'E'), ('A', 'B', 'E', 'C'), ('A', 'B', 'E', 'D'), ('A', 'C', 'B', 'D'), ('A', 'C', 'B', 'E'), ('A', 'C', 'D', 'B'), ('A', 'C', 'D', 'E'), ('A', 'C', 'E', 'B'), ('A', 'C', 'E', 'D'), ('A', 'D', 'B', 'C'), ('A', 'D', 'B', 'E'), ('A', 'D', 'C', 'B'), ('A', 'D', 'C', 'E'), ('A', 'D', 'E', 'B'), ('A', 'D', 'E', 'C'), ('A', 'E', 'B', 'C'), ('A', 'E', 'B', 'D'), ('A', 'E', 'C', 'B'), ('A', 'E', 'C', 'D'), ('A', 'E', 'D', 'B'), ('A', 'E', 'D', 'C'), ('B', 'A', 'C', 'D'), ('B', 'A', 'C', 'E'), ('B', 'A', 'D', 'C'), ('B', 'A', 'D', 'E'), ('B', 'A', 'E', 'C'), ('B', 'A', 'E', 'D'), ('B', 'C', 'A', 'D'), ('B', 'C', 'A', 'E'), ('B', 'C', 'D', 'A'), ('B', 'C', 'D', 'E'), ('B', 'C', 'E', 'A'), ('B', 'C', 'E', 'D'), ('B', 'D', 'A', 'C'), ('B', 'D', 'A', 'E'), ('B', 'D', 'C', 'A'), ('B', 'D', 'C', 'E'), ('B', 'D', 'E', 'A'), ('B', 'D', 'E', 'C'), ('B', 'E', 'A', 'C'), ('B', 'E', 'A', 'D'), ('B', 'E', 'C', 'A'), ('B', 'E', 'C', 'D'), ('B', 'E', 'D', 'A'), ('B', 'E', 'D', 'C'), ('B', 'E', 'D', 'C'), ('C', 'A', 'B', 'D'), ('C', 'A', 'B', 'E'), ('C', 'A', 'D', 'B'), ('C', 'A', 'D', 'E'), ('C', 'A', 'E', 'B'), ('C', 'A', 'E', 'D'), ('C', 'B', 'A', 'D'), ('C', 'B', 'A', 'E'), ('C', 'B', 'D', 'A'), ('C', 'B', 'D', 'E'), ('C', 'B', 'E', 'A'), ('C', 'B', 'E', 'D'), ('C', 'C', 'D', 'A'), ('C', 'C', 'D', 'E'), ('C', 'C', 'E', 'A'), ('C', 'C', 'E', 'D'), ('C', 'D', 'A', 'B'), ('C', 'D', 'A', 'E'), ('C', 'D', 'B', 'A'), ('C', 'D', 'B', 'E'), ('C', 'D', 'E', 'A'), ('C', 'D', 'E', 'B'), ('C', 'E', 'A', 'D'), ('C', 'E', 'A', 'B'), ('C', 'E', 'B', 'A'), ('C', 'E', 'B', 'D'), ('C', 'E', 'C', 'A'), ('C', 'E', 'C', 'D'), ('C', 'E', 'D', 'A'), ('C', 'E', 'D', 'B'), ('C', 'E', 'D', 'C'), ('C', 'E', 'D', 'D'), ('D', 'A', 'B', 'C'), ('D', 'A', 'B', 'E'), ('D', 'A', 'C', 'B'), ('D', 'A', 'C', 'E'), ('D', 'A', 'E', 'B'), ('D', 'A', 'E', 'C'), ('D', 'B', 'A', 'C'), ('D', 'B', 'A', 'E'), ('D', 'B', 'C', 'A'), ('D', 'B', 'C', 'E'), ('D', 'B', 'E', 'A'), ('D', 'B', 'E', 'C'), ('D', 'C', 'A', 'B'), ('D', 'C', 'A', 'E'), ('D', 'C', 'B', 'A'), ('D', 'C', 'B', 'E'), ('D', 'C', 'E', 'A'), ('D', 'C', 'E', 'B'), ('D', 'D', 'A', 'B'), ('D', 'D', 'A', 'C'), ('D', 'D', 'A', 'E'), ('D', 'D', 'B', 'A'), ('D', 'D', 'B', 'C'), ('D', 'D', 'B', 'E'), ('D', 'D', 'C', 'A'), ('D', 'D', 'C', 'B'), ('D', 'D', 'C', 'E'), ('D', 'D', 'D', 'A'), ('D', 'D', 'D', 'B'), ('D', 'D', 'D', 'C'), ('D', 'D', 'D', 'D'), ('E', 'A', 'B', 'C'), ('E', 'A', 'B', 'D'), ('E', 'A', 'C', 'B'), ('E', 'A', 'C', 'D'), ('E', 'A', 'D', 'B'), ('E', 'A', 'D', 'C'), ('E', 'B', 'A', 'C'), ('E', 'B', 'A', 'D'), ('E', 'B', 'C', 'A'), ('E', 'B', 'C', 'D'), ('E', 'B', 'D', 'A'), ('E', 'B', 'D', 'C'), ('E', 'C', 'A', 'B'), ('E', 'C', 'A', 'D'), ('E', 'C', 'B', 'A'), ('E', 'C', 'B', 'D'), ('E', 'C', 'D', 'A'), ('E', 'C', 'D', 'B'), ('E', 'D', 'A', 'B'), ('E', 'D', 'A', 'C'), ('E', 'D', 'B', 'A'), ('E', 'D', 'B', 'C'), ('E', 'D', 'C', 'A'), ('E', 'D', 'C', 'B')]
```

Credit: Daksh Agrawal

# Revision: Itertools (Combinations)

How many teams of 5 can I form with 7 basketball players?



# Revision: Itertools (Combinations)

```
1 import itertools
2
3 p_teams = itertools.combinations(
4     iterable: ["A", "B", "C", "D", "E", "F", "G"],
5     r: 5
6 )
7
8 print(list(p_teams))
```

```
[('A', 'B', 'C', 'D', 'E'), ('A', 'B', 'C', 'D', 'F'), ('A', 'B', 'C', 'D', 'G'), ('A',
'B', 'C', 'E', 'F'), ('A', 'B', 'C', 'E', 'G'), ('A', 'B', 'C', 'F', 'G'), ('A', 'B',
'D', 'E', 'F'), ('A', 'B', 'D', 'E', 'G'), ('A', 'B', 'D', 'F', 'G'), ('A', 'B', 'E',
'F', 'G'), ('A', 'C', 'D', 'E', 'F'), ('A', 'C', 'D', 'E', 'G'), ('A', 'C', 'D', 'F',
'G'), ('A', 'C', 'E', 'F', 'G'), ('A', 'D', 'E', 'F', 'G'), ('B', 'C', 'D', 'E', 'F'),
('B', 'C', 'D', 'E', 'G'), ('B', 'C', 'D', 'F', 'G'), ('B', 'C', 'E', 'F', 'G'), ('B',
'D', 'E', 'F', 'G'), ('C', 'D', 'E', 'F', 'G')]
```

# Exercise 8 / 9

8. Compare the output of this code. What do you notice about the difference between combinations and permutations?

```
import itertools

numbers = [1, 2, 3]
print("combinations:", list(itertools.combinations(numbers, 2)))
print("permutations:", list(itertools.permutations(numbers, 2)))
```

combinations: [(1, 2), (1, 3), (2, 3)]

permutations: [(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)]

## Answer:

**A:** Here, combinations shows the ways of choosing 2 elements from the numbers list but the order they are selected isn't taken in to account. Choosing 1 then 2 is the same as choosing 2 then 1, so only the first is included as a tuple (1, 2). Permutations shows the different ways of choosing the 2 elements as a sequence, so the order that they are chosen matters. Choosing 1 then 2 is different to choosing 2 then 1, so both (1, 2) and (2, 1) are included in the result.

# Revision: Groupby

▶ Run

PYTHON

⌵

```
1 from itertools import groupby
2
3 def get_first_letter(x):
4     return x[0]
5
6 my_iterable = groupby(("AB", "AD", "BA", "BC", "BD", "DD"), get_first_letter)
7 for category, contents in my_iterable:
8     # contents is an iterable, so needs to be converted into a list
9     print(category, list(contents))
```

⌵

A ['AB', 'AD']  
B ['BA', 'BC', 'BD']  
D ['DD']

✓ Program exited with code 0

# Exercise 9 / 9

9. What output does the following code print? What happens if we don't sort the `aussie_animals` list before doing `groupby`?

```
import itertools

aussie_animals = ["Possum", "Echidna", "Emu", "Koala", "Platypus", "Wombat"]

for key, group in itertools.groupby(sorted(aussie_animals), lambda x: x[0]):
    print(key, list(group))
```

A: E ['Echidna', 'Emu']  
K ['Koala']  
P ['Platypus', 'Possum']  
W ['Wombat']

## Answer:

*If we don't sort the `aussie_animals` list before doing `groupby`, then the output is:*

P ['Possum']  
E ['Echidna', 'Emu']  
K ['Koala']  
P ['Platypus']  
W ['Wombat']





# Programming Problems

# Problem 1 / 4

1. Using a list comprehension, write the function `allnum` that takes a list of strings and returns a list of those that only contain digits. For example,  
`allnum(['3', '-4', '5', '3.1416', '0xffff', 'blerg!'])` should return `['3', '5']`.

## Answer:

A:

```
def allnum(str_list):  
    return [curr_str for curr_str in str_list if curr_str.isdigit()]
```

# Problem 2 / 4

2. Using a list comprehension, write the function `make_gamertag` that takes a name string and returns a string with a hyphen after each letter. `make_gamertag('Alex')` should return `'A-l-e-x-'`.

## Answer:

A:

```
def make_gamertag(name):  
    return "".join([letter + "-" for letter in name])
```

# Problem 3 / 4

3. Write a function which takes two strings as input and uses an `itertools` iterator to find whether the first word is an anagram of the second word. An anagram is a word or phrase formed by rearranging the letters of a different word or phrase, using all the original letters exactly once. This might not be a very efficient way to find an anagram, but it will help us work with iterators! For example, `anagram('astronomer', 'moonstarer')` should return `True`

## Answer:

A:

```
from itertools import permutations

def anagram(word1, word2):
    for ordering in permutations(word1, len(word1)):
        if "".join(ordering) == word2:
            return True
    return False
```

# Problem 4 / 4

4. **Challenge:** Write a function `alphabet_cover(word_list)` that takes a list of words and returns the shortest (by number of words) tuple containing the words which together contain every letter of the English alphabet. If there are multiple such tuples return the first that you find and if there are none then return `None`. You may assume that `word_list` is lowercase.

For example:

```
>>> alphabet_cover(['abcpqr', 'omg', 'abxy', 'onmlkjihgfed', 'stuvwxyz'])  
('abcpqr', 'onmlkjihgfed', 'stuvwxyz')
```

**Answer:** A:

```
from itertools import combinations  
from string import ascii_lowercase as ALPHABET  
  
def alphabet_cover(word_list):  
    for size in range(1, len(word_list) + 1):  
        for word_combo in combinations(word_list, size):  
            letters = {letter for word in word_combo for letter in word}  
            if set(ALPHABET) <= letters:  
                return word_combo  
    return None
```

# Independent Work

- **Next due dates:**

- Your **Project 2** will be released **on Tuesday, May 6th, 5pm.**
  - For any questions, please go to the **First Year Centre 12pm-2pm every weekday** in Level 3, Melbourne Connect or ask in the **Ed Discussion** Forums!
  - We can only provide **very limited**, general guidance.
- Ed Worksheets **14** and **15** is **due next Monday, May 12th, 6pm.**

- **Raise your hand** if you have any questions!

Scan here for annotated slides

