THE UNIVERSITY OF MELBOURNE

COMP10001 - Sem 2 2024 - Week 9

Foundations of Computing

Daksh Agrawal

```
new_list = [<expression> <for iterations> <if filter>]
```

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

a = [("evelyn", 0), ("alex", 0), ("sam", 0)]

a = []

```
for name in ("....."):
    a.append((name, 0))
```

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

b = [1, 9, 25]

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

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

"PYTHON"

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

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

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

```
e = ((name, 0) for name in ("evelyn", "alex", "sam"))
```

Iterator

```
7  words = ['pencil', 'highlighter', 'paper-clip', 'ruler', 'pen']
8  # A list containing only the words that start with 'p'
9  result = [word for word in words if word[0] == "p"]
10
11 # A dictionary mapping each word to their length
12 result = {word : len(word) for word in words}
13
14 # A set of every character used in words
15 result = {letter for word in words for letter in word}
16
```

tuple

list

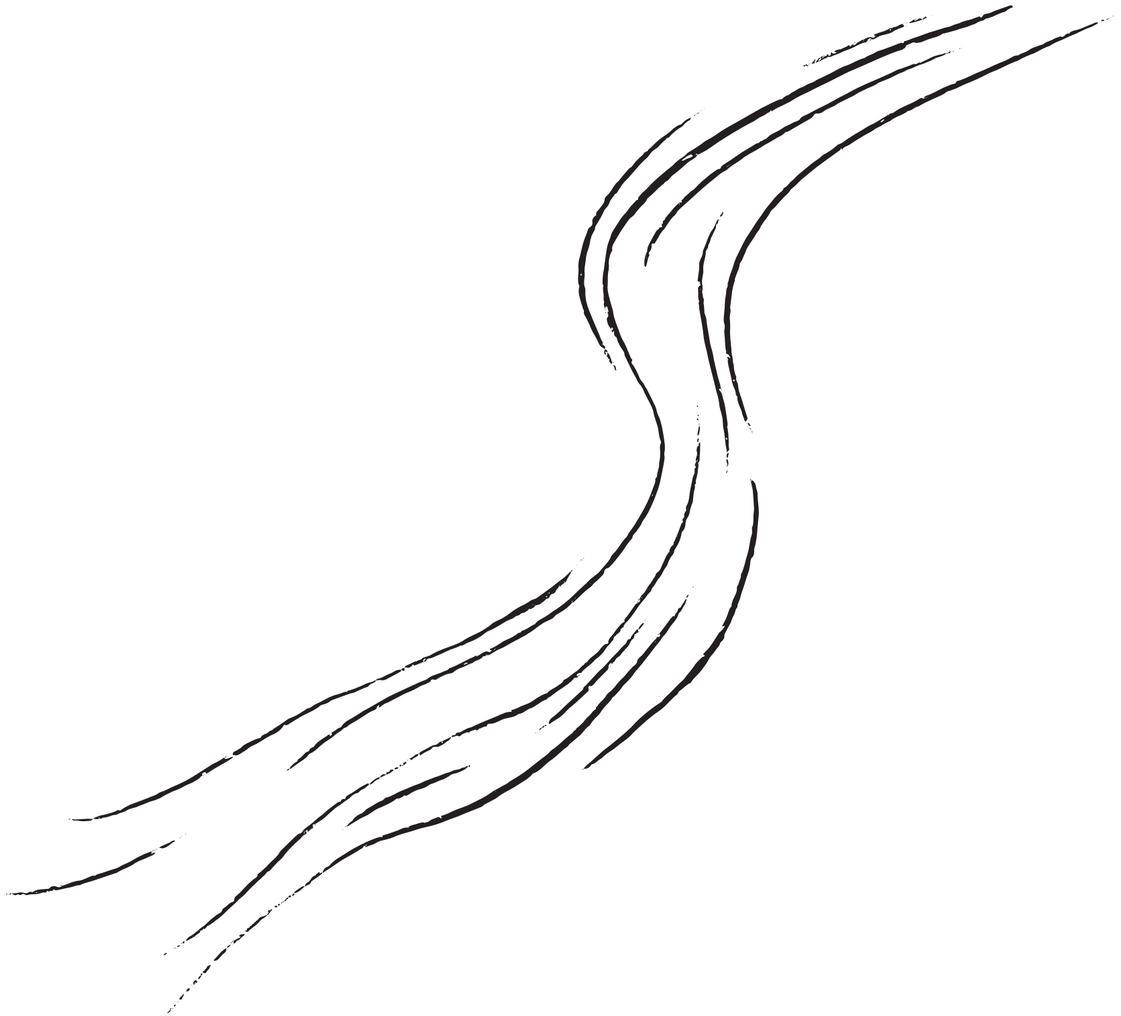
str

Iterables

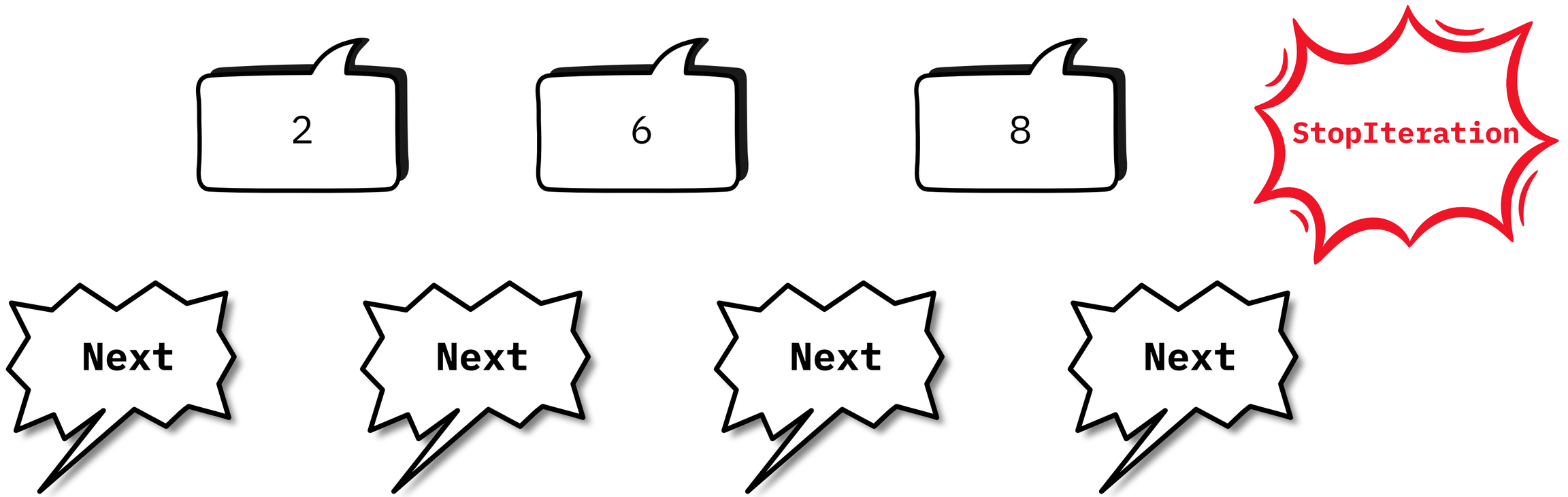
dict

set

Iterator



LIVE


2
6
8
StopIteration
Next
Next
Next
Next

```
1  it = iter([2, 4, 6])
2
3  print(next(it))
4  print(next(it))
5  print(next(it))
6  print(next(it))
```

```
2
4
6
Traceback (most recent call last):
  File "/Users/dakshagrawal/PycharmProjects/comp10001/comprehensions.py", line 6, in <module>
    print(next(it))
          ^^^^^^^^
StopIteration
```

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

```
iterable = "ABCDEFGH"
```

iterator = iter(iterable)
first = next(iterator)
second = next(iterator)

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

product

cycle

permutations

itertools

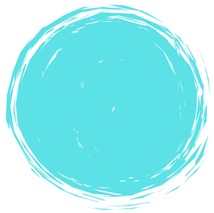
combinations

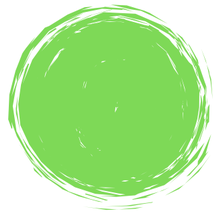
...

groupby

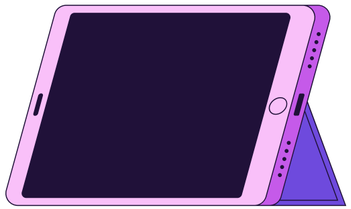
```python
import itertools

beatboxer = itertools.cycle(['boots', 'and', 'cats', 'and'])
for count in range(39):
    print(next(beatboxer))
```

Products





Products



128GB

256GB

512GB

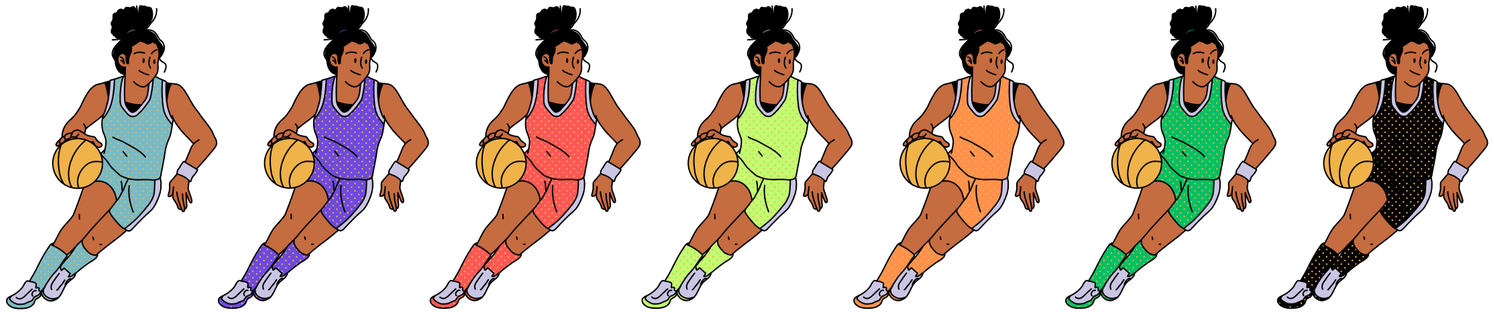
```python
import itertools

all_products = itertools.product(
    ['laptop', 'phone', 'tablet'],
    ['Y', 'R', 'P', 'B', 'G'],
    [128, 256, 512]
)

print(list(all_products))
```

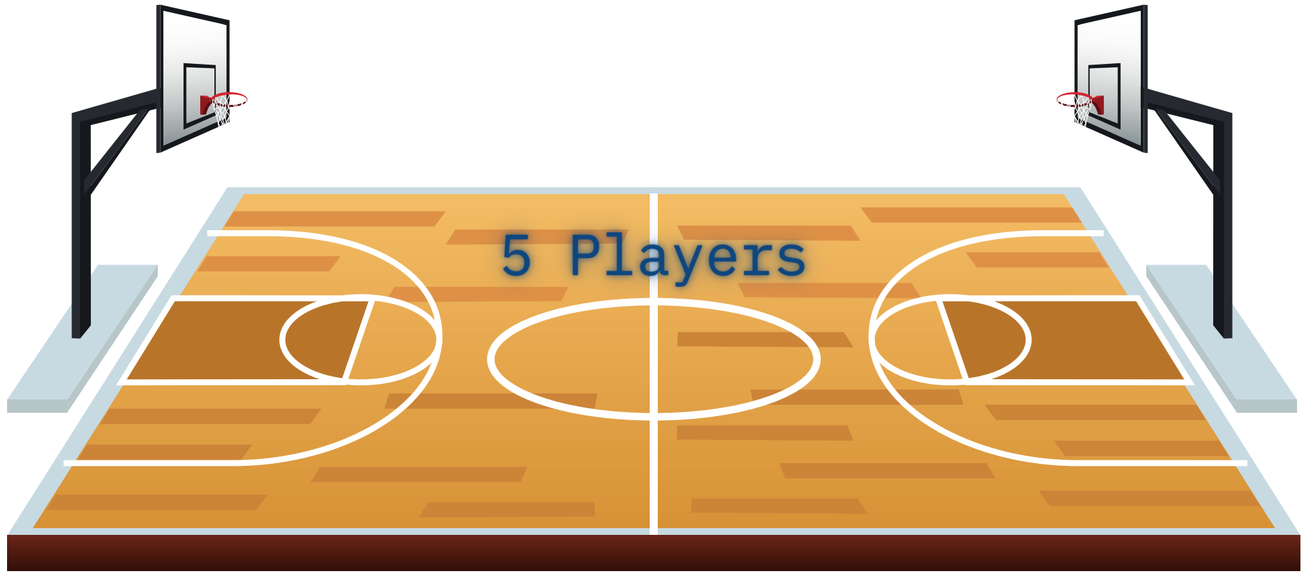
```
[('laptop', 'Y', 128), ('laptop', 'Y', 256), ('laptop', 'Y', 512), ('laptop', 'R', 128), ('laptop', 'R', 256), ('laptop', 'R', 512), ('laptop', 'P', 128), ('laptop', 'P', 256), ('laptop', 'P', 512), ('laptop', 'B', 128), ('laptop', 'B', 256), ('laptop', 'B', 512), ('laptop', 'G', 128), ('laptop',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

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:

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

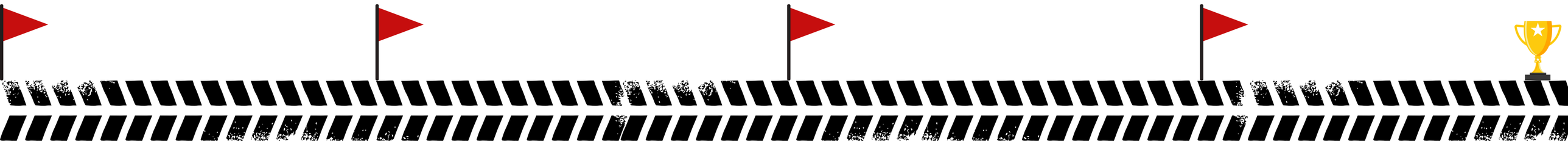
```
import itertools
for animal, place in itertools.product(animals, places):
    print(f"{animal} in {place}")
```




5 Players

```python
import itertools

p_teams = itertools.combinations(
    iterable: ["A", "B", "C", "D", "E", "F", "G"],
    r: 5
)

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')]
```



```python
import itertools

p_teams = itertools.permutations(
    iterable: ["A", "B", "C", "D", "E"],
    r: 4
)

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

What output does the following code print? What happens if we don’t sort the **aussie_animals** list before doing **groupby**?

```
1  import itertools
2  aussie_animals = ["Possum", "Echidna", "Emu", "Koala", "Platypus", "Wombat"]
3  for key, group in itertools.groupby(sorted(aussie_animals), lambda x: x[0]):
4      print(key, list(group))
5
```

Using a list comprehension, rewrite the function allnum that takes a list of strings, and returns a list of those that exclusively contain digits.

allnum(['3', '-4', '5', '3.1416', '0xfff', 'blerg!']) should return ['3', '5'].

```
def allnum(l):
    return [word for word in l if word.isdigit()]
```

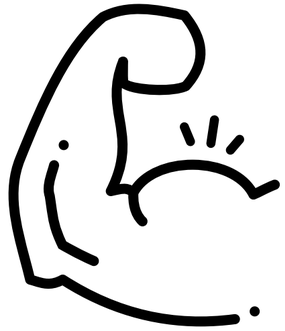
Write a **make_gamertag** function that takes a name string and returns a string with a hyphen after each letter, but this time, using a list comprehension. **make_gamertag('Alex')** should return **'A-l-e-x-'**.

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

    return "-".join(name) + "-"
```

"A-l-e-x"

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. This might not be a very efficient way to find an anagram but it will help us work with iterators!
anagram('astronomer', 'moonstarer') should return **True**

Write a function **alphabet_cover(word_list)** that takes a list of words and returns the shortest (by number of words) tuple of words 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.



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


MAROON 5
NEW SINGLE
MAPS