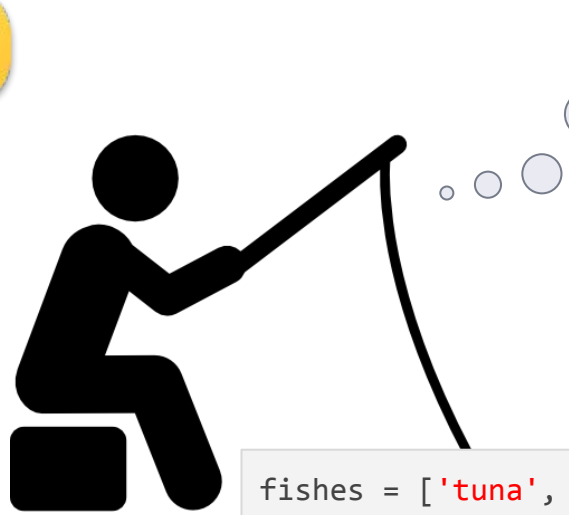



```
fishes = ['tuna', 'squid', 'seabass']
```

Accessing Lists

Table of Contents

```
fruit[1]
```

```
fruit = ['Apple', 'Orange', 'Banana']
```

Indexing a `list`

```
list()
```

Indexing a `list`

- The formula syntax 👇

```
list_name[index no]
```

Indexing a `list`

- To access or use the elements of a `list`, we can use index numbers of the `list` enclosed by square brackets.

```
list_name[index no]
```

```
word = ['h', 'a', 'p', 'p', 'y']
print(word[1])
```

Indexing a `list`

- To access or use the elements of a `list`, we can use index numbers of the `list` enclosed by square brackets.

```
list_name[index no]
```

```
word = ['h', 'a', 'p', 'p', 'y']
print(word[1])
```

```
a
```

Indexing a `list` (review the pre-class)

- Here is the pre-class example of indexing a `list` :

```
1  colors = ['red', 'purple', 'blue', 'yellow', 'green']
2  print(colors[2])  # If we start at zero,
3  # the second element will be 'blue'.
4
```

Indexing a `list` (review the pre-class)

- Here is an example of indexing a `list` :


```
1  colors = ['red', 'purple', 'blue', 'yellow', 'green']
2  print(colors[2])  # If we start at zero,
3  # the second element will be 'blue'.
4
```

```
1  blue
2
```

Indexing a `list` (review the pre-class)

- Here is another pre-class example of indexing a nested `list`.

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2
3  city_list = []
4  city_list.append(city) # we have created a nested list
5
6  print(city_list)
7
```

Indexing a `list` (review the pre-class)

- Here is another pre-class example of indexing a nested `list`.

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2
3  city_list = []
4  city_list.append(city) # we have created a nested list
5
6  print(city_list)
7
```

```
1  [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2
```

How many items do the `city` list have?

Students, write your response!

Indexing a `list` (review the pre-class)

- Here is another example of indexing a nested `list`.

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2
3  city_list = []
4  city_list.append(city) # we have created a nested list
5
6  print(city_list)
7
```

⚠ `city_list` has only one item.

```
1  [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2
```

Tips :

- If you notice that `city_list` has double square brackets.

Indexing a `list`

- Let's access `city_list`'s first and the only element.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0]) # access to first and only element
3
```

Indexing a `list`

- Let's access `city_list`'s first and the only element.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0]) # access to first and only element
3
```

```
1  ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2
```

Indexing a `list`

- Let's access `city_list`'s first and the only element.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0]) # access to first and only element
3
```

```
1  ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2
```

- The output of the syntax `city_list[0]` is a `list` type. So we can still access its elements.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0][2])
3
```

⚠ it is also a `list`

What is the output? Try to figure out in your mind...

Students choose an option

Indexing a `list`

- Let's access `city_list`'s first and the only element.

```
city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
print(city_list[0]) # access to first and only element

```

```
['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']

```

- The output of the syntax `city_list[0]` is a `list` type. So we can still access its elements.

```
city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
print(city_list[0][2])

```

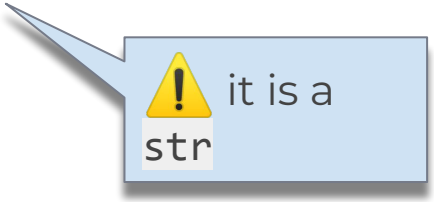
```
Istanbul

```

Indexing a `list`

- The output of the syntax `city_list[0][2]` is a `str` type. So we can still access its elements.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0][2][3])
3
```



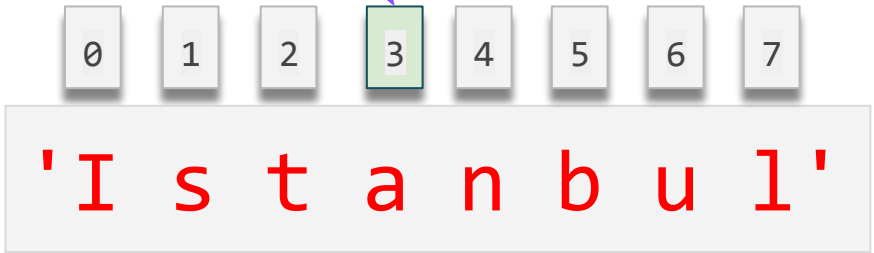
What is the output? Try to figure out in your mind...

Indexing a `list`

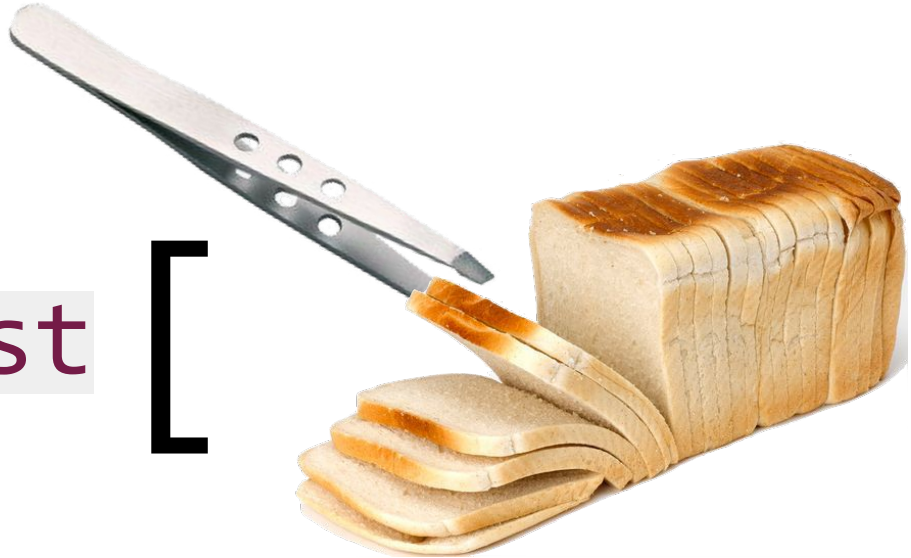
- The output of the syntax `city_list[0][2]` is a `str` type. So we can still access its elements.

```
1  city_list = [['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']]
2  print(city_list[0][2][3])
3
```

```
1  a
2
```



Slicing a `list`



What do you understand from slicing a `list`?

Type your understanding from pre-class content.

EXCAVATE AND EXPAND

Students, write your response!

Slicing a `list`

- The formula syntax 👇

```
list_name = [start:stop:step]
```

From '`start`' to '`stop-1`', by '`step`'.

Slicing a `list`

- Consider this simple example :

```
1  even_numbers = [2, 4, 6, 8, 10, 12, 14]
2  print(even_numbers[2:5])
3
4
```

What is the output? Try to figure out in your mind...

Students, write your response!

Slicing a `list`

- Consider this simple example :

```
1  even_numbers = [2, 4, 6, 8, 10, 12, 14]
2  print(even_numbers[2:5])
3
4
```

Output

```
[6, 8, 10]
```

⚠ The type of the output is also a `list`.

Slicing a `list`

- Consider this simple example :

```
1  even_numbers = [2, 4, 6, 8, 10, 12, 14]
2  print(even_numbers[2:5])
3
4
```

Tips :

- Slicing is just similar to indexing. The difference is adding **colon** or **colons** in square brackets.

```
[6, 8, 10]
```

range() function

- Returns a list of arithmetic progressions.
  - As we stated before, the formula syntax of the `range()` function is :

```
range(start, stop, step)
```

parameters

```
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(1, 11))
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> list(range(0, 30, 5))
[0, 5, 10, 15, 20, 25]
```

Slicing a `list`

- Task :
  - Create a `list` of numbers from **1** to **10** using `range()` function and select **odd** ones by **slicing** method and then print the result.

Review the function

- `range(start,stop,step)` function returns an object that produces a sequence of integers from `start` (including) to `stop` (excluding) by `step`.

Slicing a `list`

- The code can be like :

```
1  odd_numbers = list(range(11))
2
3  print(odd_numbers)
4  print(odd_numbers[1:11:2])
5
6
7
```

Output

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 3, 5, 7, 9]
```

Slicing a `list`

- Usage options of slicing are as follows :

- `my_list[:]`: returns the full copy of the sequence
- `my_list[start:]` : returns elements from `start` to the end element
- `my_list[:stop]` : returns element from the 1st element to `stop-1`
- `my_list[::step]` : returns each element with a given `step`

Slicing a `list` (review the pre-class)

- Consider this pre-class example :

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2
3  print(animals[:])  # all elements of the list
4
```

Slicing a `list` (review the pre-class)

- Consider this simple example :

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2
3  print(animals[:])  # all elements of the list
4
```

```
1  ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2
```

`print(animals) = print(animals[:])`

These syntaxes give the same output.

Slicing a `list` (review the pre-class)

- Slicing options :

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[3:])
3
```

Slicing a `list` (review the pre-class)

- The following example slices the `animals` starts at index=3 to the end.

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[3:])
3
```

```
1  ['wolf', 'rabbit', 'deer', 'giraffe']
2
```

Slicing a `list` (review the pre-class)

- Slicing options :

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[:5])
3
```

Slicing a `list` (review the pre-class)

- The following example slices the `animals` starts at index=0 to the index=4.

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[:5])
3
```

```
1  ['elephant', 'bear', 'fox', 'wolf', 'rabbit']
2
```

Slicing a `list` (review the pre-class)

- Slicing options :

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[::2])
3
```

Slicing a `list` (review the pre-class)

- This example slices animals starts at index=0 to the end with 2 step.

```
1  animals = ['elephant', 'bear', 'fox', 'wolf', 'rabbit', 'deer', 'giraffe']
2  print(animals[::2])
3
```

```
1  ['elephant', 'fox', 'rabbit', 'giraffe']
2
```

Slicing a `list`

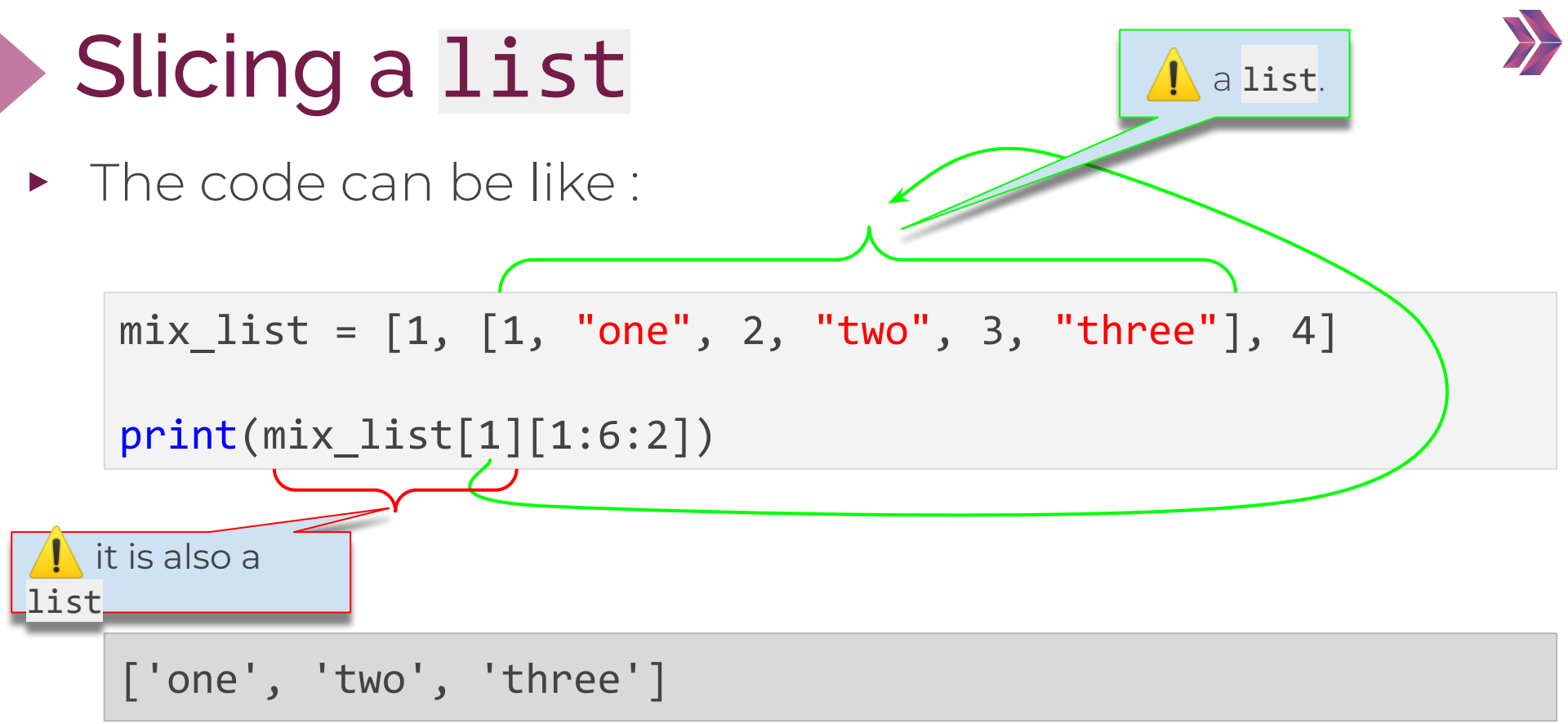
- Task :
  - Select and print the **string typed numbers** from the following `list` using *indexing* and *slicing* methods.
  - Your code must consist of a **single line**.

```
mix_list = [1, [1, "one", 2, "two", 3, "three"], 4]
```

Students, write your response!

Slicing a `list`

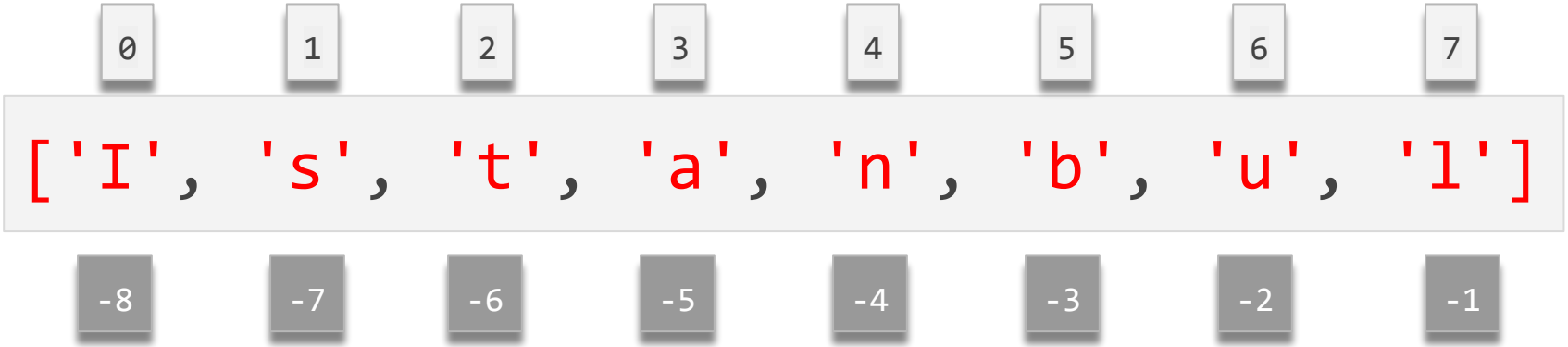
- The code can be like :



Negative Indexing & Slicing

Negative Indexing & Slicing

- Sample of the negative indices sequence



Negative Indexing & Slicing(review)

- Take a look at this pre-class example of negative indexing.

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2  print(city[-4])
3
```

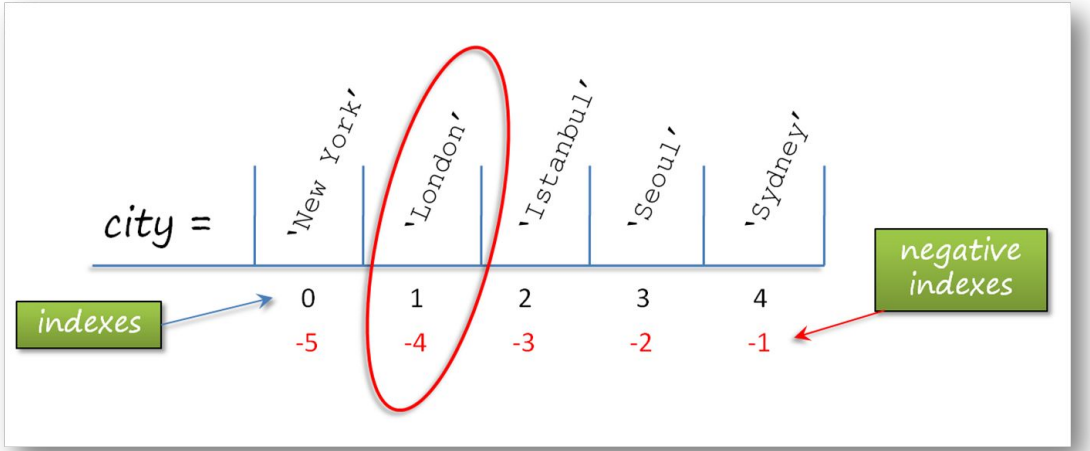
What is the output? Try to figure out in your mind...

Negative Indexing & Slicing(review)

- The output is.

```
1  city = ['New York', 'London', 'Istanbul', 'Seoul', 'Sydney']
2  print(city[-4])
3
```

```
1  London
2
```



Negative Indexing & Slicing(review)

- Now, let's consider this pre-class example of negative slicing.

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[-3:])
3
```

What is the output? Try to figure out in your mind...

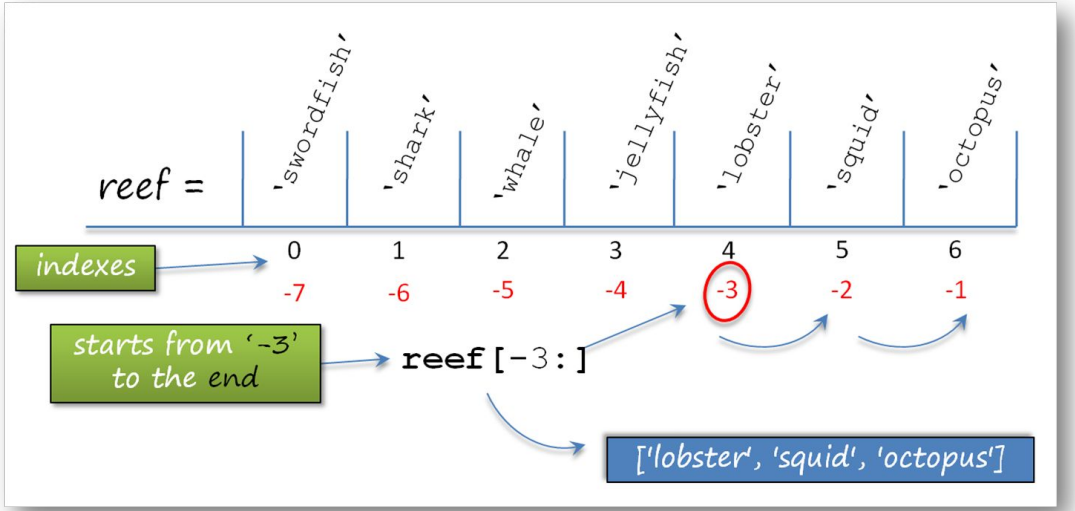
Students, write your response!

Negative Indexing & Slicing(review)

- The output is :

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[-3:])
3
```

```
1  ['lobster', 'squid', 'octopus']
2
```



Negative Indexing & Slicing(review)

- Here's another pre-class example of negative slicing.

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[:-3])
3
```

What is the output? Try to figure out in your mind...

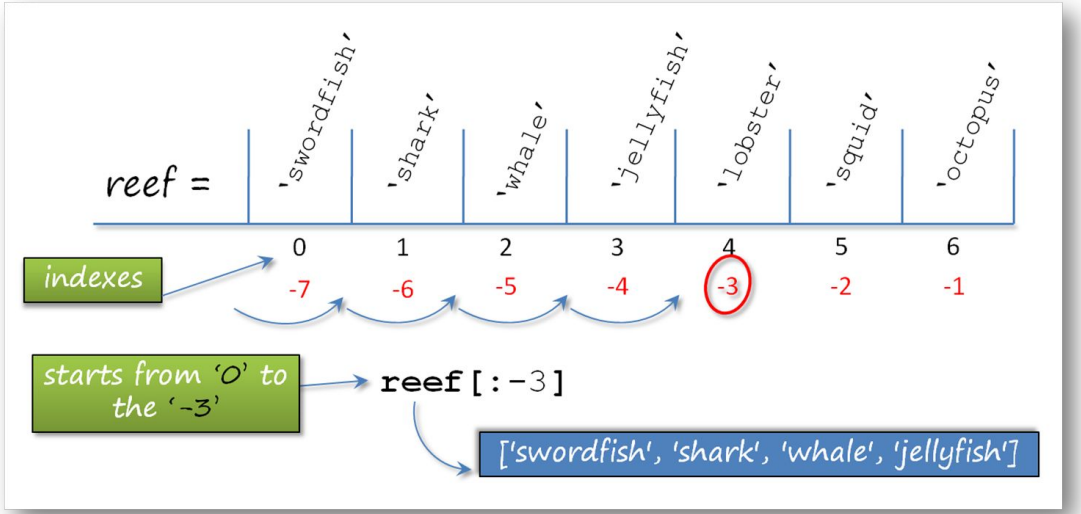
Students, write your response!

Negative Indexing & Slicing(review)

- The output is :

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[:-3])
3
```

```
1  ['swordfish', 'shark', 'whale', 'jellyfish']
2
```



Negative Indexing & Slicing(review)

- Let's see negative **step**ping in the pre-class content :

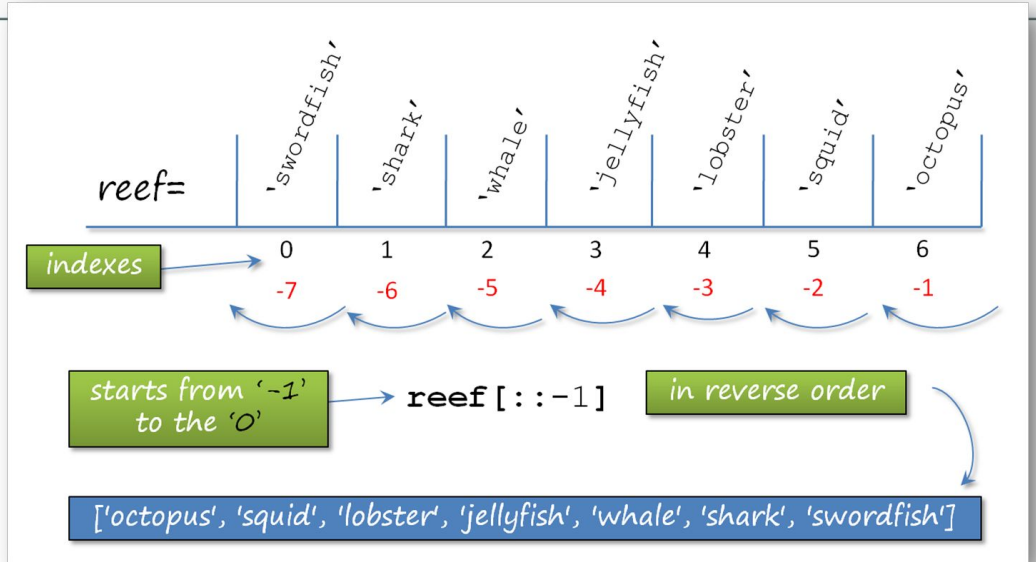
```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[::-1])  # we have produced the reverse of the list
3
```

Negative Indexing & Slicing(review)

- The output is :

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[::-1])  # we have produced the reverse of the list
3
```

```
1  ['octopus', 'squid', 'lobster', 'jellyfish', 'whale', 'shark', 'swordfish']
2
```



Negative Indexing & Slicing(review)

- Here's another example of negative `step`ping.

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[::-2])
3
```

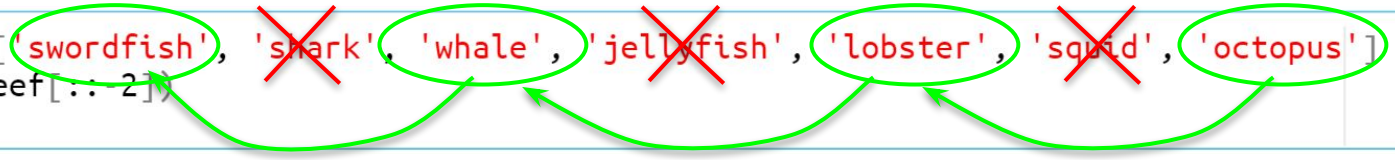
What is the output? Try to figure out in your mind...

Students, write your response!

Negative Indexing & Slicing(review)

- Here's another example of negative **stepping**.

```
1  reef = ['swordfish', 'shark', 'whale', 'jellyfish', 'lobster', 'squid', 'octopus']
2  print(reef[::-2])
3
```



```
['octopus', 'lobster', 'whale', 'swordfish']
```

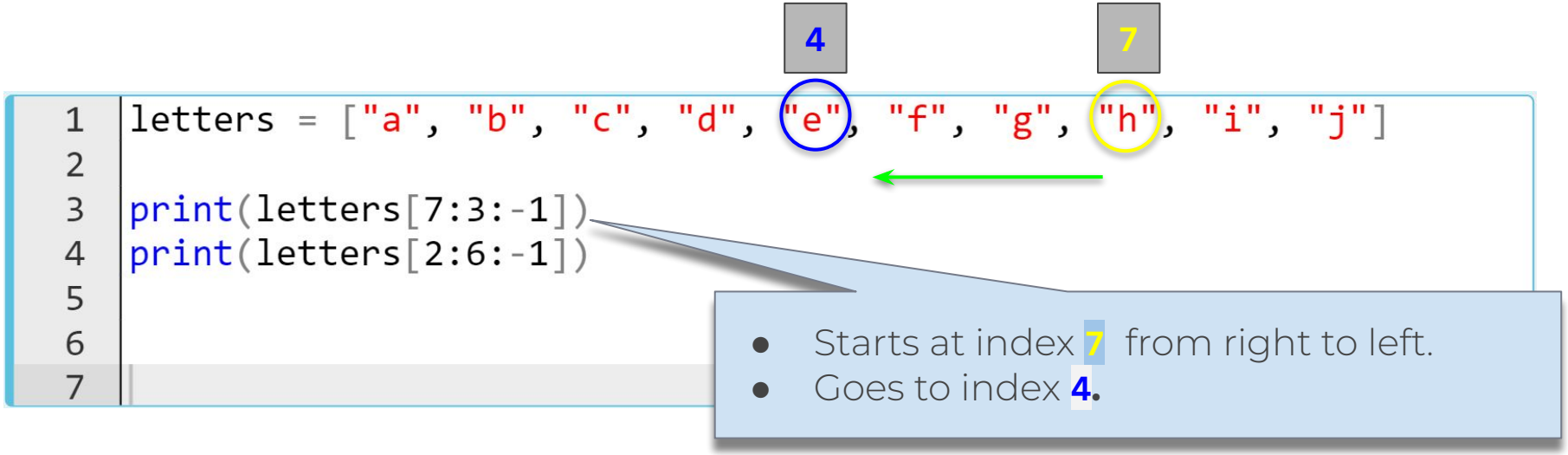
Negative Indexing & Slicing

Tips :

- If you choose negative step with the start and end indexes together, those should be used accordingly, that is, the `end` index should be less than the `start` index.

```
1  letters = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j"]
2
3  print(letters[7:3:-1])
4  print(letters[2:6:-1])
5
6
7
```

Negative Indexing & Slicing




```
['h', 'g', 'f', 'e']
[]
```

Negative Indexing & Slicing


```
1  letters = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j"]
2
3  print(letters[7:3:-1])
4  print(letters[2:6:-1])
5
6
7
```

- Starts at index **2** from right to left.
- No way to reach index **6**.
- So, the output is an empty `list`.

```
['h', 'g', 'f', 'e']
[]
```

Did you fully understand Python lists?

YES
(Evet, tam anladım)

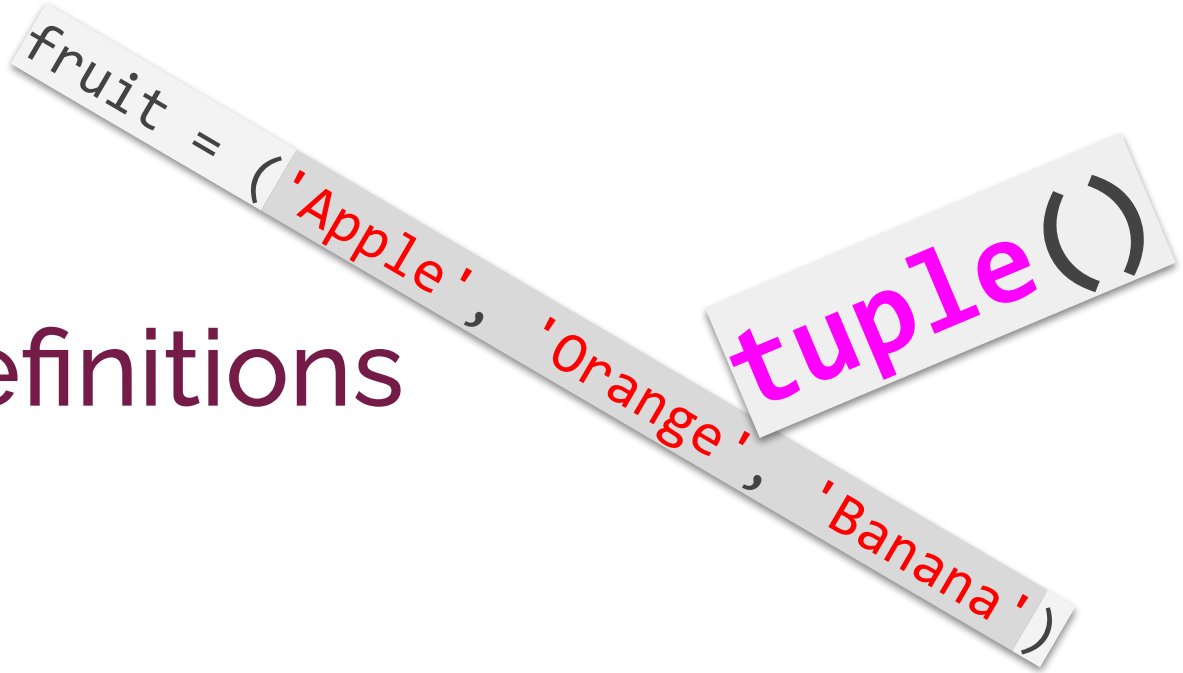
Not a lot
(Tam değil!)

NO
(Hiç anlamadım)

Tuples

Table of Contents

Definitions



What can you say about the tuples:

Write at least 2 sentences.

Students, write your response!

Definitions

Definitions

- Your data is safe.

Definitions

- Lists vs Tuples

Creating a `tuple`

Creating a `tuple`

- We have two basic ways to create a tuple.

```
()
tuple()
```

Creating a `tuple`

- A `tuple` can be created by enclosing values, separated by commas, in parentheses 👉 `()`.
- Another way to create a `tuple` is to call the `tuple()` function.

- `()`
- `tuple()`

```
tuple_1 = ('h', 'a', 'p', 'p', 'y')
word = 'happy'
tuple_2 = tuple(word)
print(tuple_1)
print(tuple_2)
```

What is the output? Try to figure out in your mind...

Creating a `tuple`

- A `tuple` can be created by enclosing values, separated by commas, in parentheses 👉 `()`.
- Another way to create a `tuple` is to call the `tuple()` function.

- `()`
- `tuple()`

```
tuple_1 = ('h', 'a', 'p', 'p', 'y')
word = 'happy'
tuple_2 = tuple(word)
print(tuple_1)
print(tuple_2)
```

⚠ an iterable object can be converted into a `tuple`

```
('h', 'a', 'p', 'p', 'y')
('h', 'a', 'p', 'p', 'y')
```

Creating a `tuple` (review the pre-class)

Here is an example of creating an empty `tuple`:

```
1  empty_tuple = ()
2  print(type(empty_tuple))
3
```

Creating a `tuple` (review the pre-class)

Here is an example of creating an empty `tuple`:

```
1  empty_tuple = ()
2  print(type(empty_tuple))
3
```

```
1  <class 'tuple'>
2
```

Creating a `tuple`

- Take a look at the following example about creating a `tuple`:

```
1  my_tuple = ("Solar")
2
3  print(my_tuple, type(my_tuple), sep="\n")
4
5
6
```

What is the output? Try to figure out in your mind...

Students, write your response!

Creating a `tuple`

- Single element tuple :

```
1  my_tuple = ("Solar")
2
3  print(my_tuple, type(my_tuple), sep="\n")
4
5
6
```

Output

```
Solar
<class 'str'>
```

Creating a `tuple`

- If you want to create a single element `tuple`, an error will probably rise, unless you do not use a **comma**:

```
1  my_tuple = ("Solar",)
2
3  print(my_tuple, type(my_tuple), sep="\n")
4
5
6
```

⚠️ comma makes it `tuple` type.

Output

```
('Solar',)
<class 'tuple'>
```

Creating a `tuple` (review the pre-class)

- Without parenthesis : 👇

```
1  solar = "Earth", "Venus", "Uranus"
2
3  print(solar, type(solar), sep="\n")
4
5
6
```

Creating a `tuple` (review the pre-class)

- Another way of creating a tuple :

```
1  solar = "Earth", "Venus", "Uranus"
2
3  print(solar, type(solar), sep="\n")
4
5
6
```

⚠ Items separated with a **comma** without parentheses are accepted as `tuple` type by Python.

Output

```
('Earth', 'Venus', 'Uranus')
<class 'tuple'>
```

Creating a `tuple`

- `list` to `tuple`, `tuple` to `list`

```
1  my_tuple=(1, 4, 3, 4, 5, 6, 7, 4)
2
3  my_list = list(my_tuple)
4
5  print(type(my_list), my_list)
6
```

Creating a `tuple`

- `list` to `tuple`, `tuple` to `list`

```
1  my_tuple=(1, 4, 3, 4, 5, 6, 7, 4)
2
3  my_list = list(my_tuple)
4
5  print(type(my_list), my_list)
6
```

```
1  <class 'list'> [1, 4, 3, 4, 5, 6, 7, 4]
2
```

Creating a `tuple`

- `list` to `tuple`, `tuple` to `list`

```
1  my_tuple=(1, 4, 3, 4, 5, 6, 7, 4)
2
3  my_list = list(my_tuple)
4
5  print(type(my_list), my_list)
6
```

```
1  <class 'list'> [1, 4, 3, 4, 5, 6, 7, 4]
2
```

```
1  my_list = [1, 4, 3, 4, 5, 6, 7, 4]
2
3  my_tuple = tuple(my_list)
4
5  print(type(my_tuple), my_tuple)
6
```

Creating a `tuple`

- `list` to `tuple`, `tuple` to `list`

```
1  my_tuple=(1, 4, 3, 4, 5, 6, 7, 4)
2
3  my_list = list(my_tuple)
4
5  print(type(my_list), my_list)
6
```

```
1  <class 'list'> [1, 4, 3, 4, 5, 6, 7, 4]
2
```

```
1  my_list = [1, 4, 3, 4, 5, 6, 7, 4]
2
3  my_tuple = tuple(my_list)
4
5  print(type(my_tuple), my_tuple)
6
```

```
1  <class 'tuple'> (1, 4, 3, 4, 5, 6, 7, 4)
2
```

Creating a `tuple`

- Creating a tuple with `tuple()` function

```
1  mountain = tuple('Alps')
2  print(mountain)
3
```

Creating a `tuple`

- Creating a tuple with `tuple()` function

```
1  mountain = tuple('Alps')
2  print(mountain)
3
```

```
1  ('A', 'l', 'p', 's')
2
```

Creating a `tuple`

- Take a look at the following example :

```
tuple_1 = 'h', 'a', 'p', 'p', 'y'
tuple_2 = 1, 3, 5
print(tuple_1)
print(type(tuple_1))
print(tuple_2)
```

What is the output? Try to figure out in your mind...

Students, write your response!

Creating a `tuple`

- Considering the parentheses: As we mentioned before, There is another and not so often used way to create a `tuple`. Take a look at the following example :

```
tuple_1 = 'h', 'a', 'p', 'p', 'y'
tuple_2 = 1, 3, 5
print(tuple_1)
print(type(tuple_1))
print(tuple_2)
```

⚠ There is no parenthesis

```
('h', 'a', 'p', 'p', 'y')
<class 'tuple'>
(1, 3, 5)
```

Refresh your mind with this interview question

Difference Between List and Tuple?

Try to write at least two things

Students, write your response!

How can We Use a Tuple?

In one minute, write the usage of `tuples`..

Students, write your response!

How can We Use a `tuple`?

- (..Continued) (review of the pre-class)
  - Just like the **`lists`**, the **`tuples`** support indexing :

```
1  even_no = (0, 2, 4)
2  print(even_no[0])
3  print(even_no[1])
4  print(even_no[2])
5  print(even_no[3])
6
```

How can We Use a `tuple`?

- (..Continued)(review of the pre-class)
  - Just like the **`lists`**, the **`tuples`** support indexing :

```
1  even_no = (0, 2, 4)
2  print(even_no[0])
3  print(even_no[1])
4  print(even_no[2])
5  print(even_no[3])
6
```

```
1  0
2  2
3  4
4  ---------------------------------------------------------------------------
5  print(even_no[3]) : IndexError: tuple index out of range
6
```

How can We Use a `tuple`?

- (..Continued)(review of the pre-class)
  - `tuple` is **immutable**.

```
1  city_list = ['Tokyo', 'Istanbul', 'Moskow', 'Dublin']
2
3  city_tuple = tuple(city_list)
4
5  city_tuple[0] = 'New York'  # you can't assign a value
6
```

How can We Use a `tuple`?

- (..Continued)(review of the pre-class)
  - And one of the most important differences of **`tuple`**s from **`list`**s is that **`tuple`** object does not support item assignment. Yes, because **`tuple`** is immutable.

```
1  city_list = ['Tokyo', 'Istanbul', 'Moskow', 'Dublin']
2
3  city_tuple = tuple(city_list)
4
5  city_tuple[0] = 'New York'  # you can't assign a value
6
```

```
1  ---------------------------------------------------------------------------
2  TypeError: 'tuple' object does not support item assignment
3
```

Using Tuples

- **Task** :
  - Let's access, select and print the string 'six" from the following tuple. 👇

```
1  mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))
2
```

Using Tuples

- **The code should be like this : 👇**

```
1  mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))
2
3  str_six = mix_tuple[2][2][0]
4
5  print(str_six)
6
7
```

Using Tuples

- **Task** :
  - What is the output ? 👇

```
1  mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))
2
3  str_six = mix_tuple[2][1:3]
4
5  print(str_six, type(str_six), sep="\n")
6
7
```

Students, write your response!

Using Tuples

- **The output :** 👇

```
1  mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))
2
3  str_six = mix_tuple[2][1:3]
4
5  print(str_six, type(str_six), sep="\n")
6
7
```

Try to figure out how the output can be like that?

Output

```
['two', ('six', 6)]
<class 'list'>
```

Using Tuples

- **Task** :
  - Access and print the **last** item and **its type** of the following tuple using **negative indexing** method : 👇

```
1  mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))
2
```

Using Tuples

- **The code should be like :** 👇

```
1  mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))
2
3  last = mix_tuple[-1]
4
5  print(last, type(last), sep="\n")
6
7
```

Try to figure out how the output can be like that?

Output

```
(5, 'fair')
<class 'tuple'>
```

Using Tuples

- **Task** :
  - Let's access, select and print the "fair" of the following tuple. 👇 Use **two options** which consisting of *normal* and *negative* indexing methods.

```
1 mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))
2
```

Using Tuples

- **The code should be like :** 👇

```
1  mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))
2
3  option_1 = mix_tuple[3][1]
4  option_2 = mix_tuple[-1][1]
5
6  print(option_1, option_2, sep = "\n")
7
8
```

Output

```
fair
fair
```

Refresh your mind with this interview question

Benefits of Immutability?

Try to write at least two things

Students, write your response!