Python Data Structures

Lists and Tuple, Dictionary, Sets

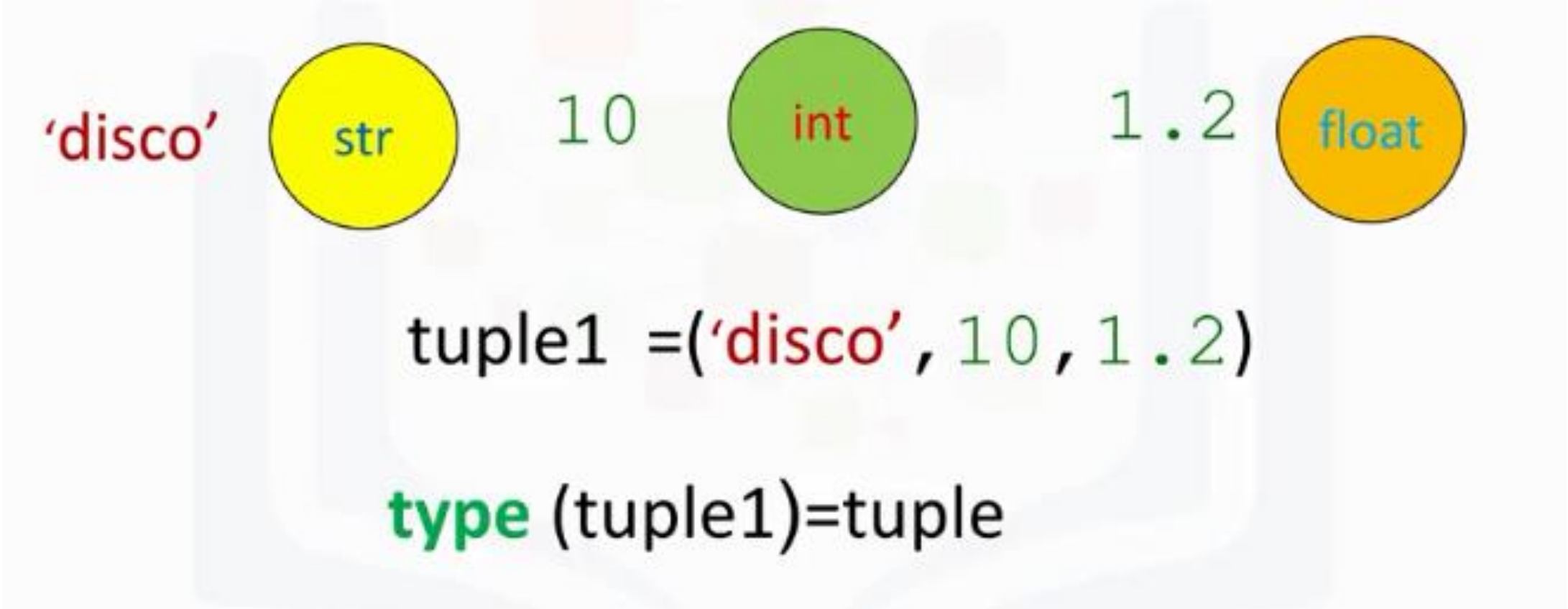
Dr. Zia Ul Rehman

Tuples

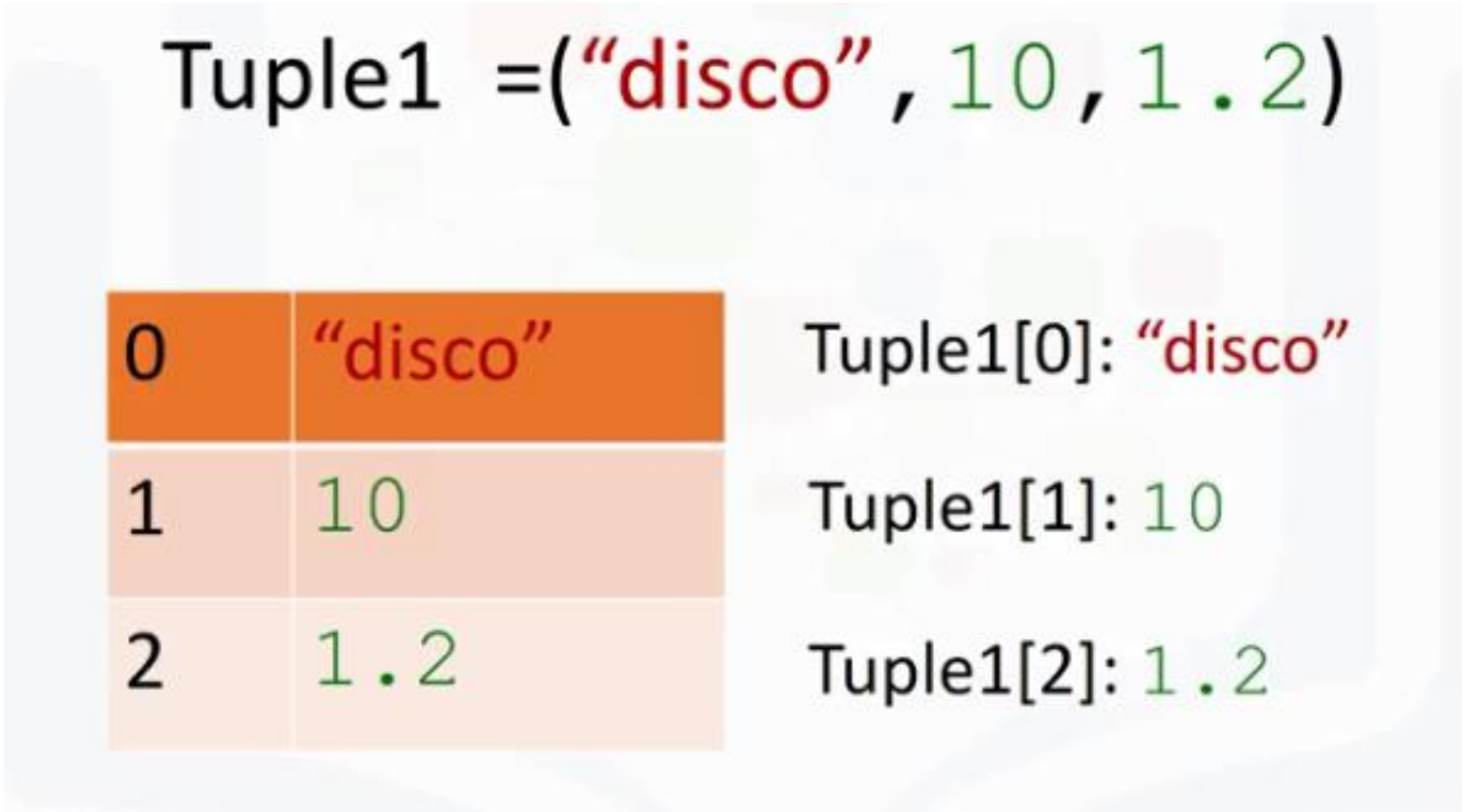
- Tuples are an ordered sequence.
- Here is a Tuple “Ratings”
- Tuples are written as comma-separated elements within parentheses

- Ratings = (10, 9, 6, 5, 10, 8, 9, 6, 2)

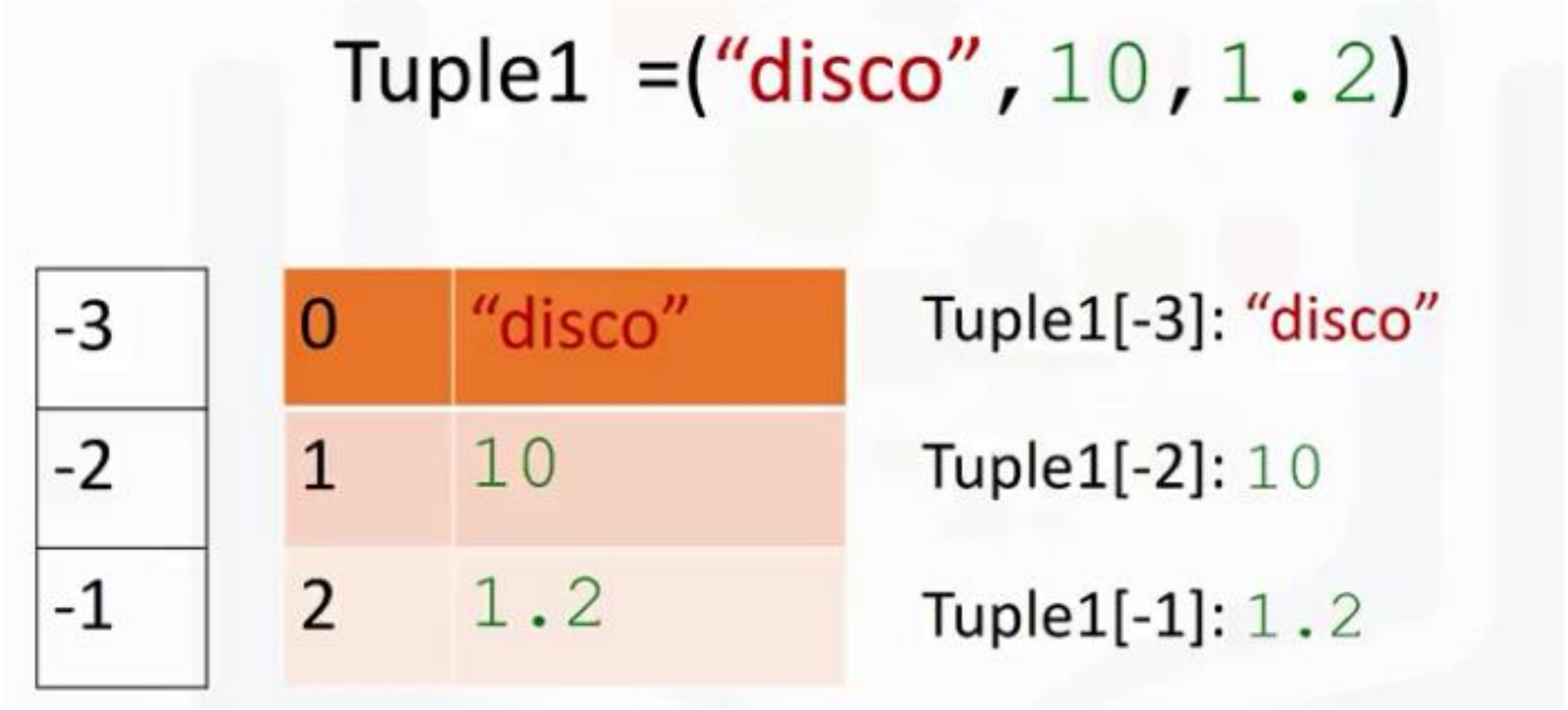
Tuples


'disco'
str
10
int
1.2
float
tuple1 =('disco',10,1.2)
type (tuple1)=tuple

Tuples


Tuple1 =("disco",10,1.2)
0
"disco"
1
10
2
1.2
Tuple1[0]: "disco"
Tuple1[1]: 10
Tuple1[2]: 1.2

Tuples


Tuple1 =("disco",10,1.2)
-3
-2
-1
0
"disco"
1
10
2
1.2
Tuple1[-3]: "disco"
Tuple1[-2]: 10
Tuple1[-1]: 1.2

Tuples

- Consider the following tuple:

```
1  say_what=('say',' what', 'you', 'will')
```

- What is the result of the following **say_what[-1]**

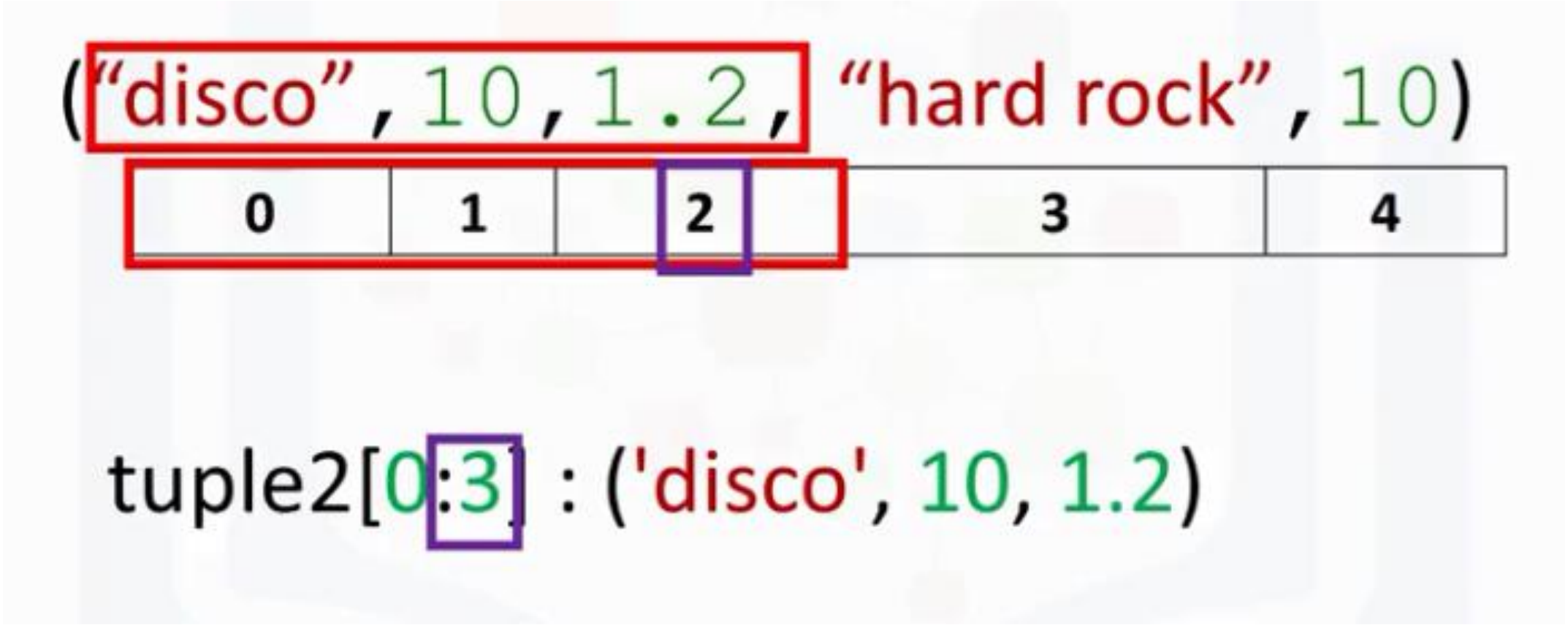
Concatenate Tuples

```
("disco",10,1.2)
```

```
tuple2 = tuple1 + ("hard rock",10)

("disco",10,1.2, "hard rock",10)
```

0	1	2	3	4

Tuples: Slicing



Tuples: Slicing

- Consider the following tuple **A=(1,2,3,4,5)**, what is the result of the following: **A[1:4]**:

Tuples: Slicing

```
len(("disco",10,1.2, "hard rock",10))=5
```

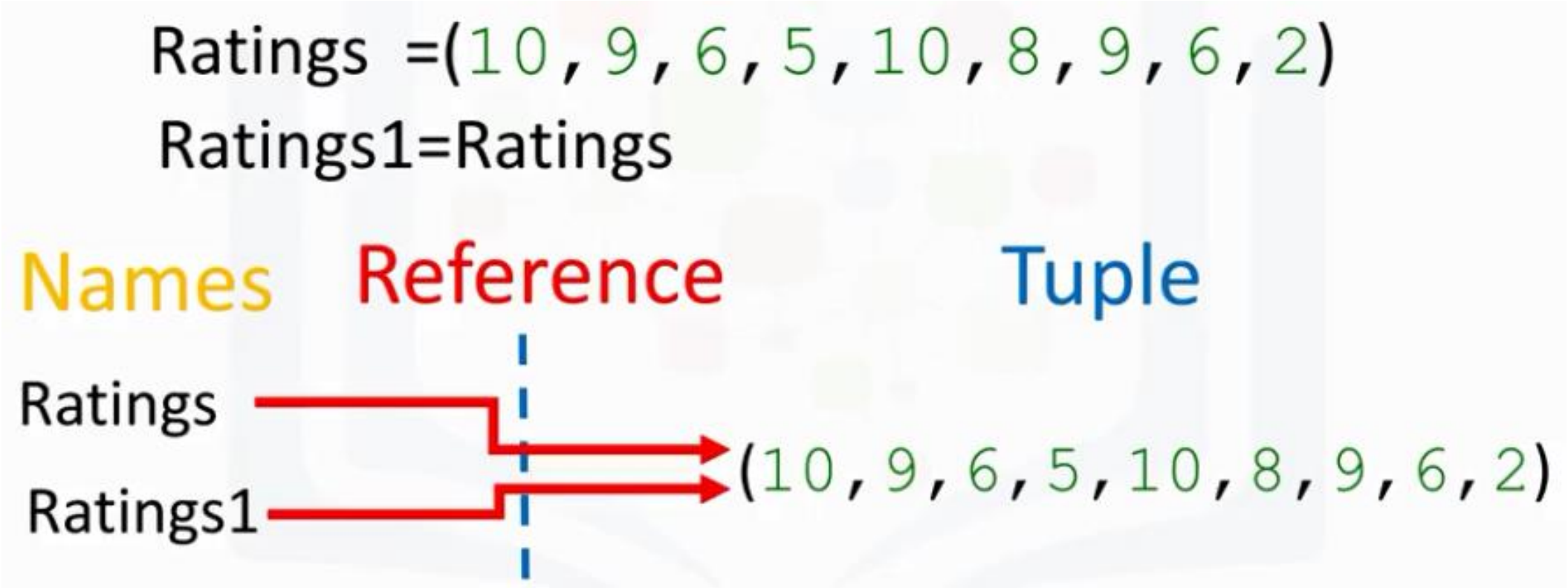
0	1	2	3	4

1	2	3	4	5

Tuples

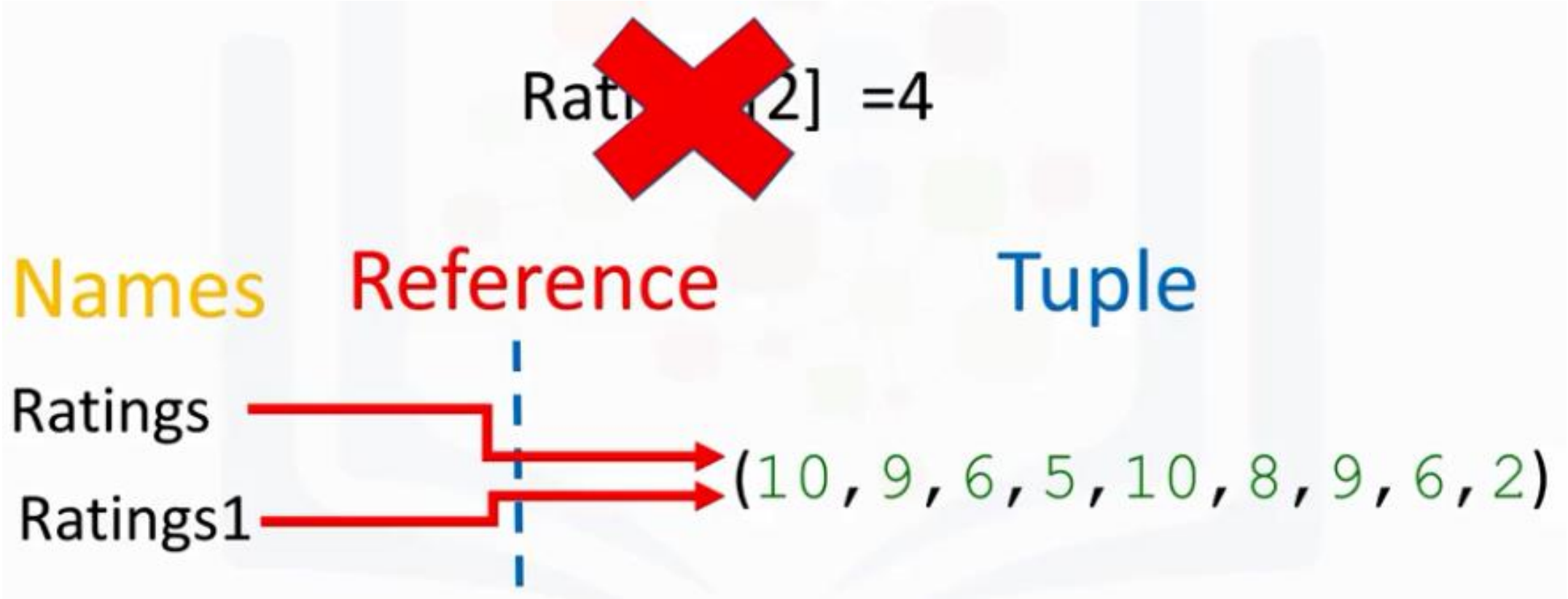
- Consider the following tuple **A=(1,2,3,4,5)**, what is the result of the following: **len(A)**

Tuples: Immutable

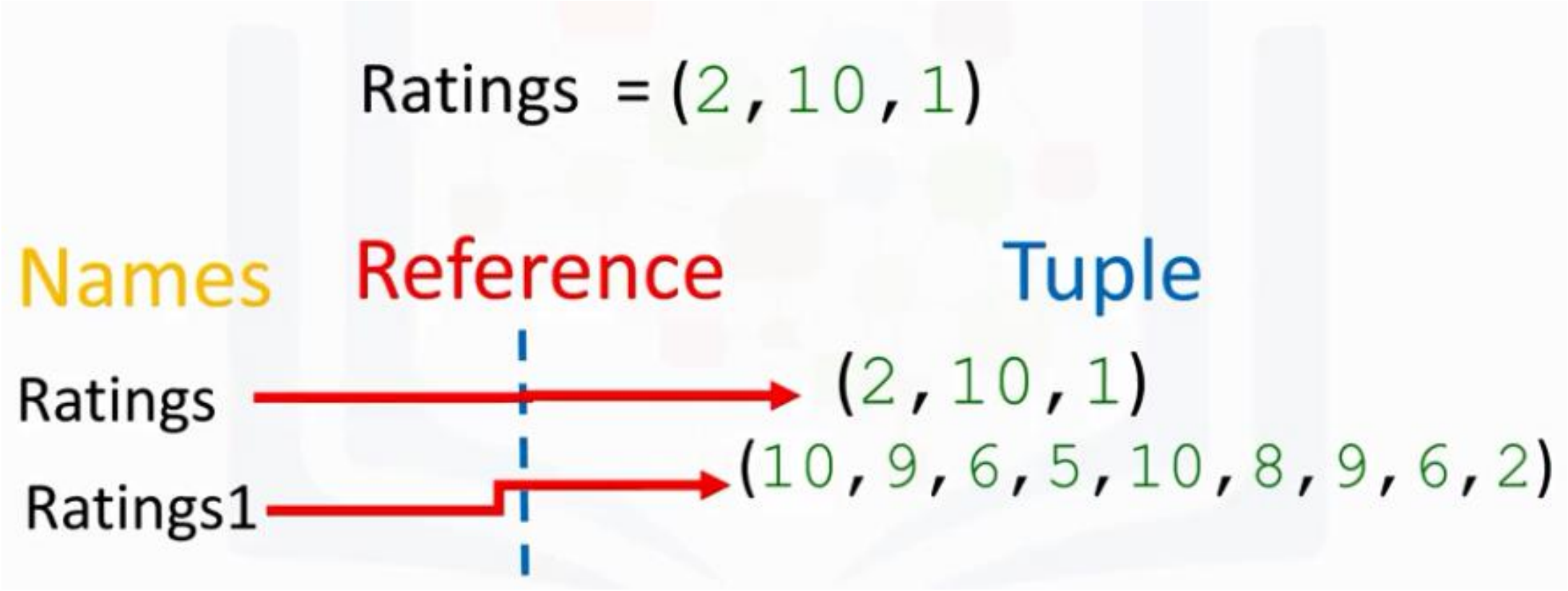


Tuples: Immutable

Tuples: Immutable



Tuples: Immutable

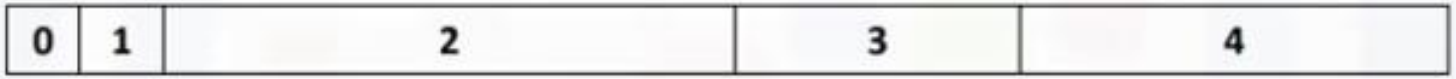


Tuples: Immutable

```
Ratings =(10,9,6,5,10,8,9,6,2)

RatingsSorted=sorted(Ratings)

(2, 5, 6, 6, 8, 9, 9, 10, 10)
```

Tuples: Nesting

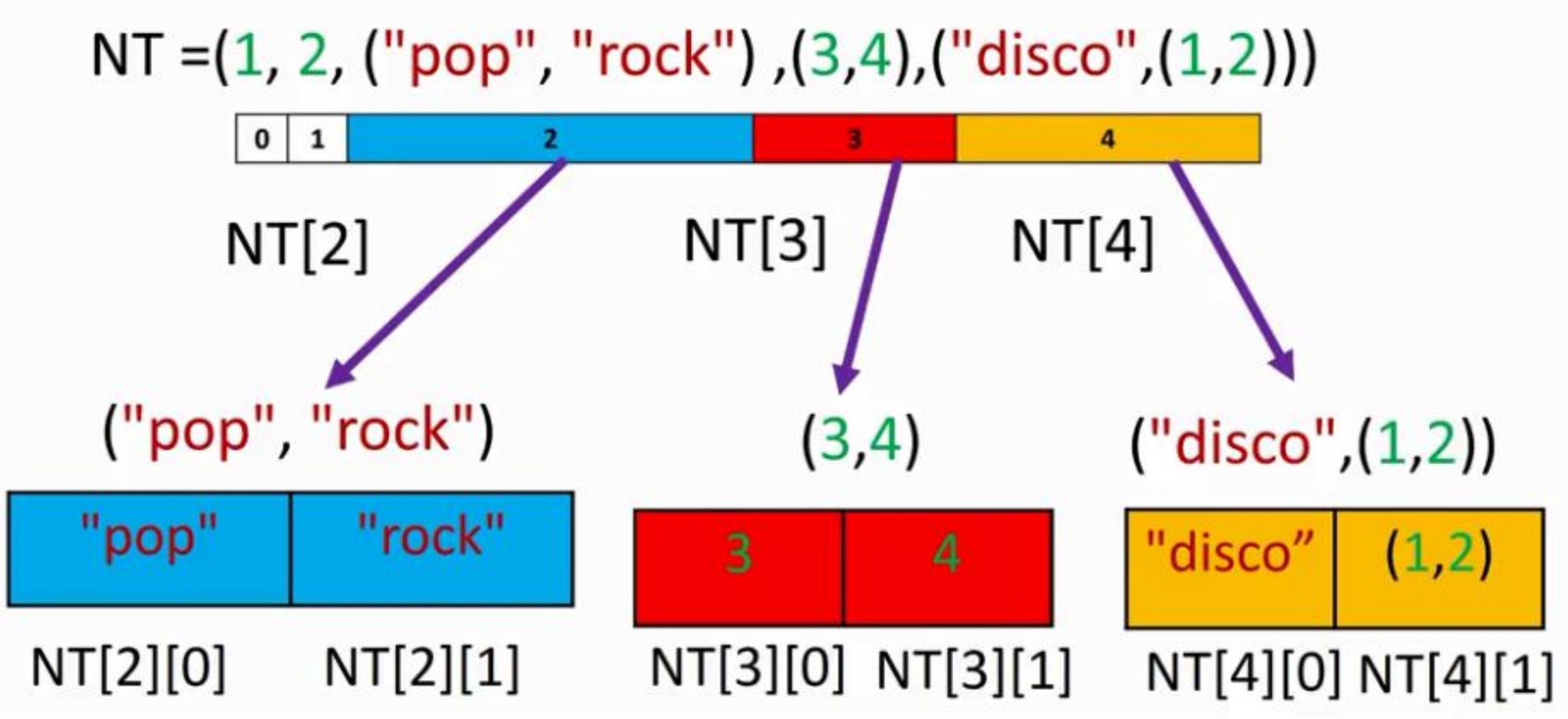
NT =(1, 2, ("pop", "rock") ,(3,4),("disco",(1,2)))



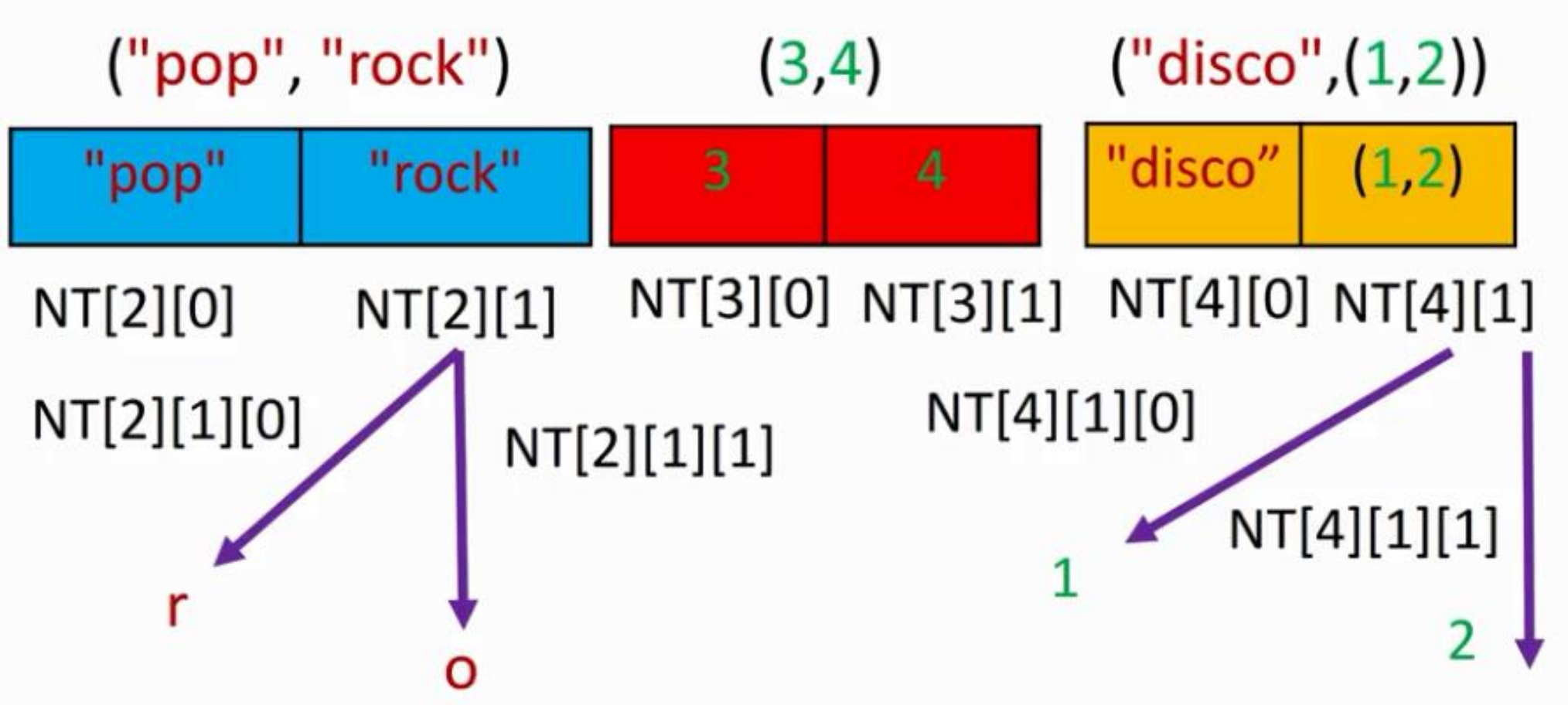
NT[2]: ("pop", "rock") [1] =“rock”

NT[2] [1] =“rock”

Tuples: Nesting



Tuples: Nesting



Lists

- Lists are also an ordered sequence.
- Here is a list L.
- A list is represented with square brackets.
- List mutable

```
L =["Michael Jackson", 10.1,1982]
```

Lists

- Lists can contain strings, floats, integers.
- We can nest other lists.
- We also nest tuples and other data structures

```
["Michael Jackson", 10.1,1982,[1,2],('A',1)]
```

Lists

- Consider the following list **B=[1,2,[3,'a'],[4,'b']]**, what is the result of the following: **B[3][1]**

Lists

```
L =["Michael Jackson", 10.1,1982]
```

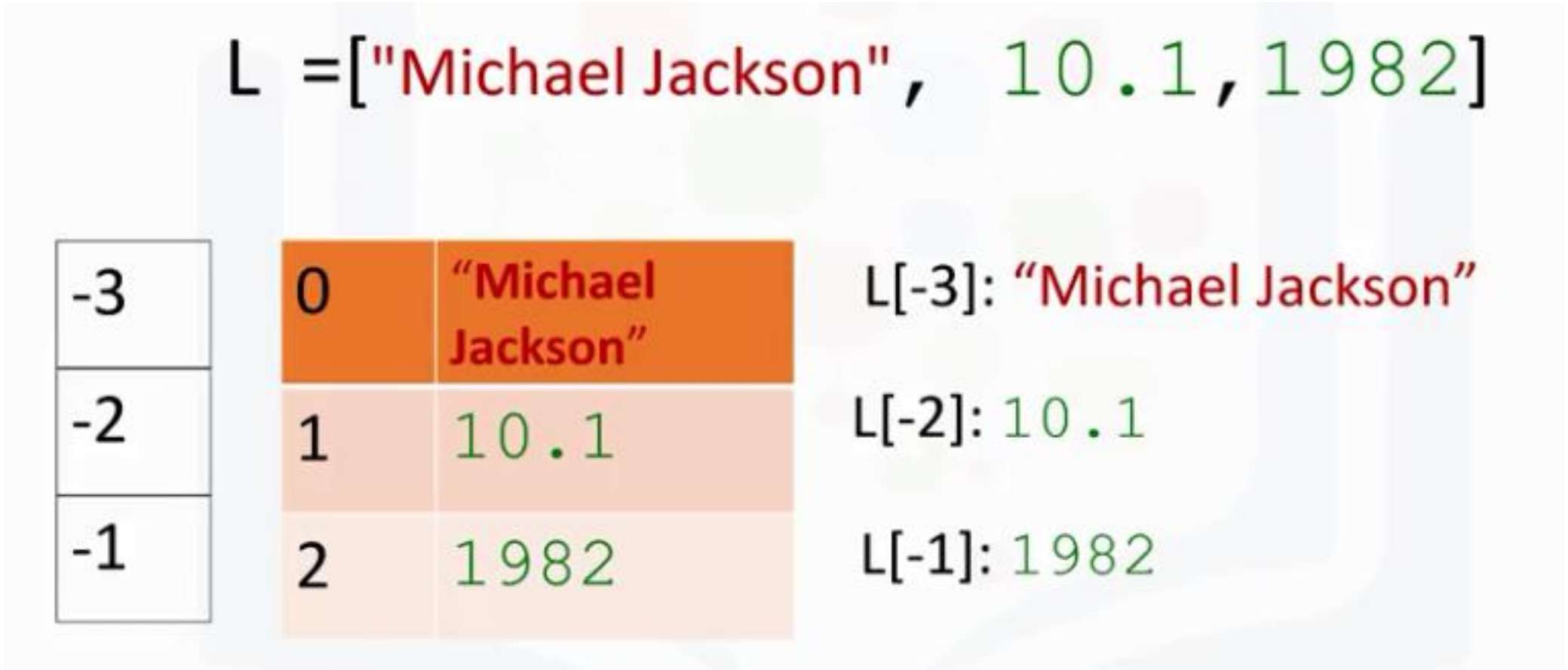
0	"Michael Jackson"
1	10.1
2	1982

L[0]: "Michael Jackson"

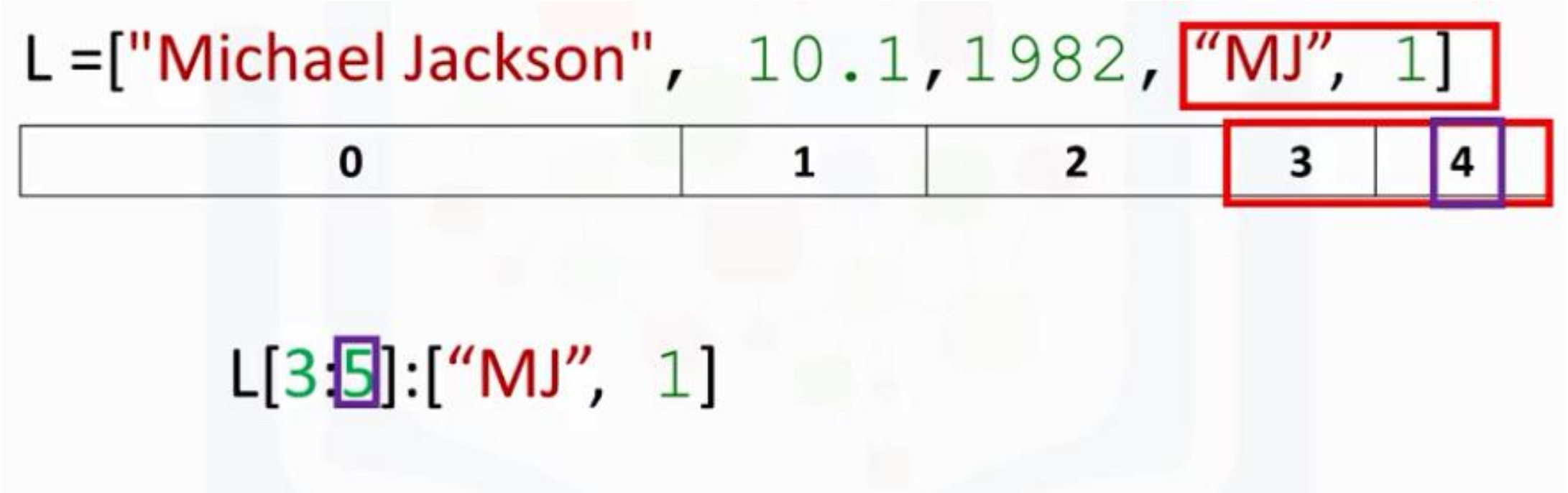
L[1]: 10.1

L[2]: 1982

Lists

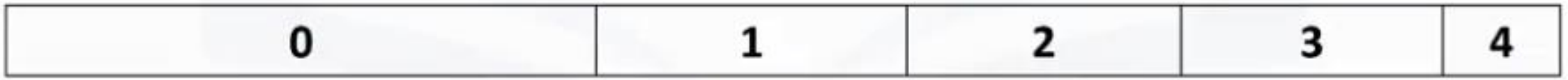

L =["Michael Jackson", 10.1,1982]
-3
-2
-1
0
"Michael Jackson"
1
10.1
2
1982
L[-3]: "Michael Jackson"
L[-2]: 10.1
L[-1]: 1982

Lists: Slicing



Concatenating Lists

```
L=["Michael Jackson", 10.1,1982]

L1 = L+[“pop”,10]

L1=[“Michael Jackson”, 10.1,1982,“pop”,10]
```



Lists

- What is the result of the following operation

- [1,2,3]+[1,1,1]

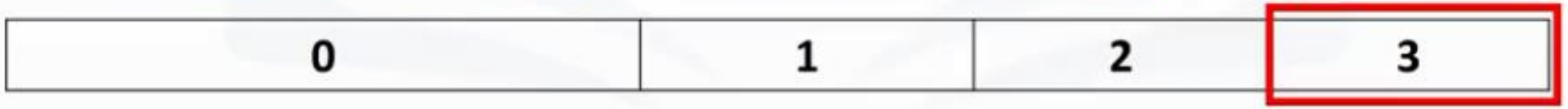
Lists: Mutable (method extend)

```
L=["Michael Jackson", 10.1,1982]

L.extend(["pop",10])

L=["Michael Jackson", 10.1,1982, "pop",10]
```

0	1	2	3	4

Lists: Mutable (method append)

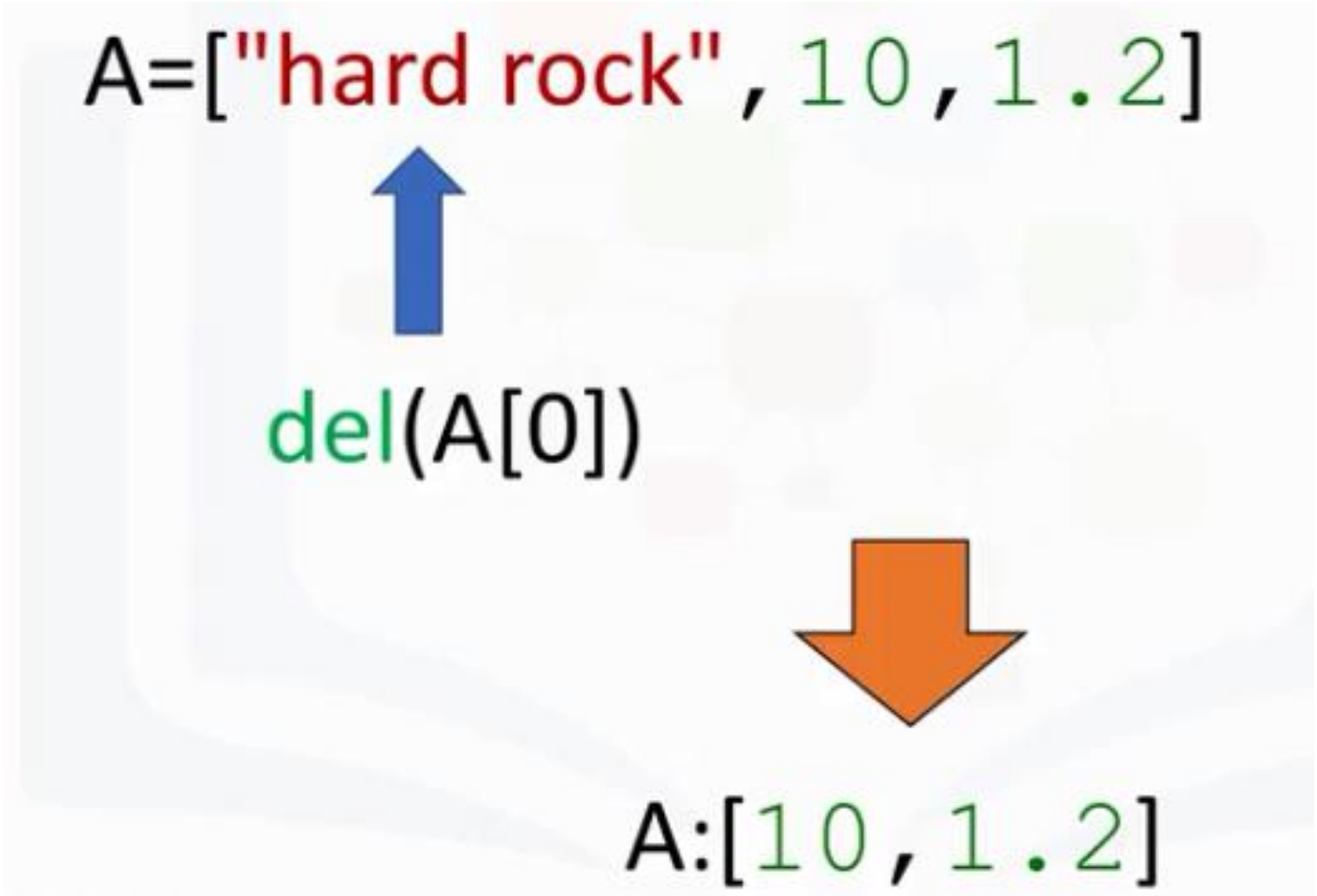
```
L=["Michael Jackson", 10.1,1982]

L.append (["pop",10])

L=["Michael Jackson", 10.1,1982, ["pop",10]]
```



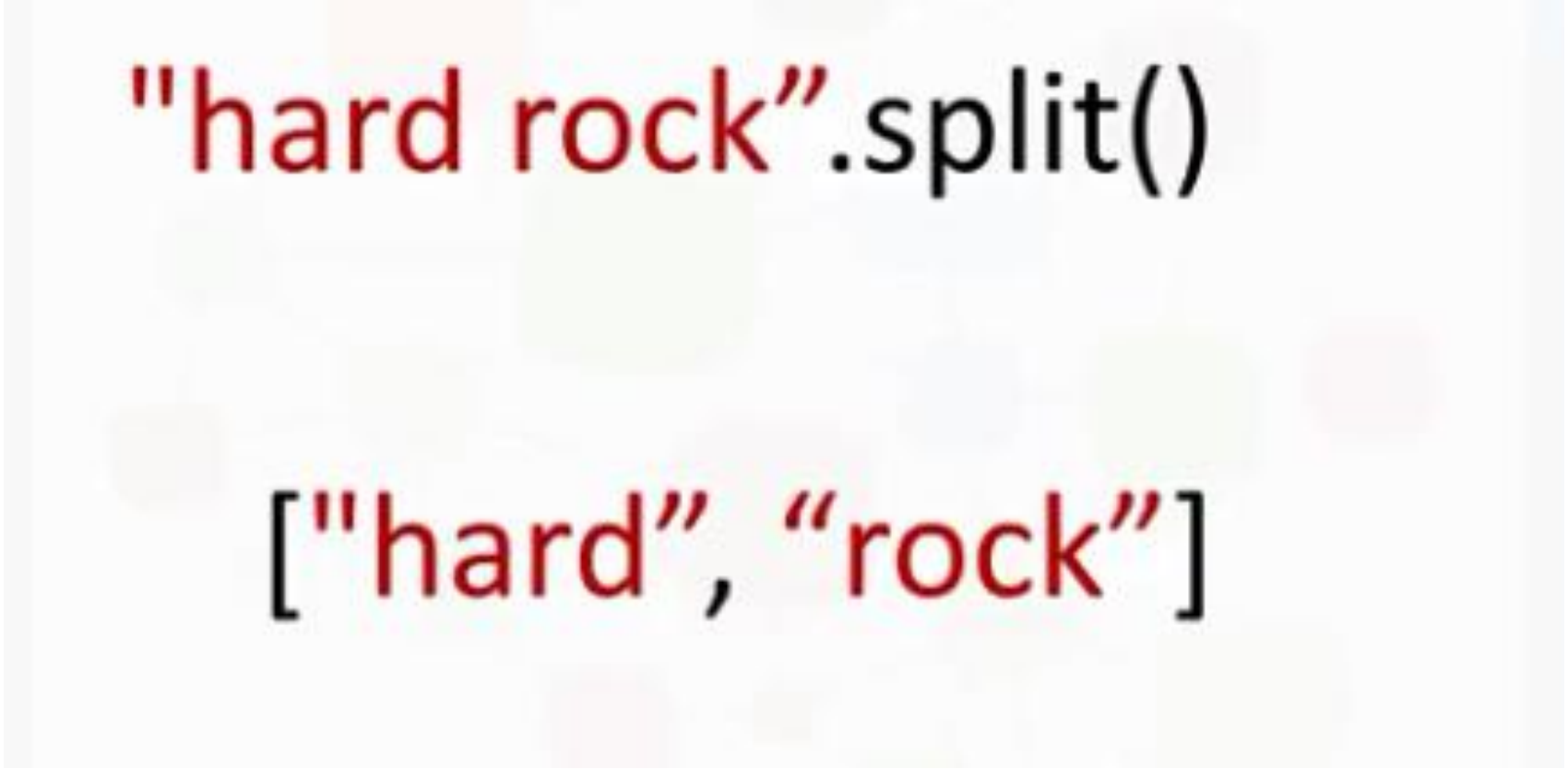
Lists: Mutable (we can change list element)

```
A=[“disco”,10,1.2]

A[0]=“hard rock”

A=["hard rock",10,1.2]
```

Lists: Mutable (we can delete list element)



Convert String to List

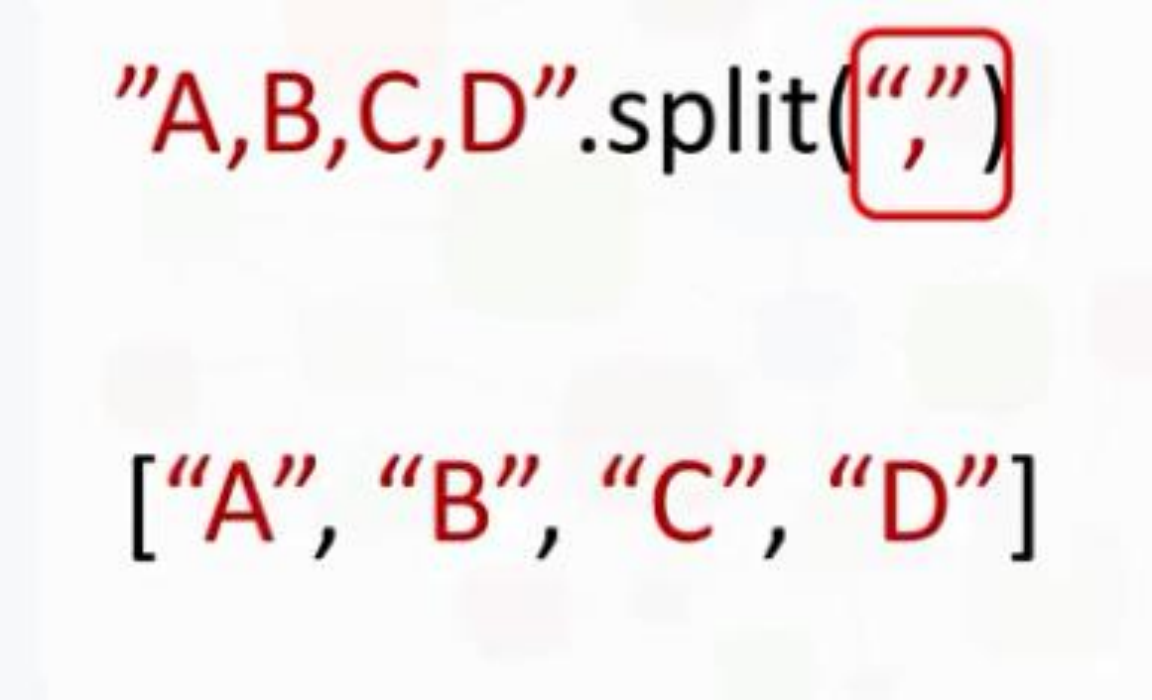


Convert String to List

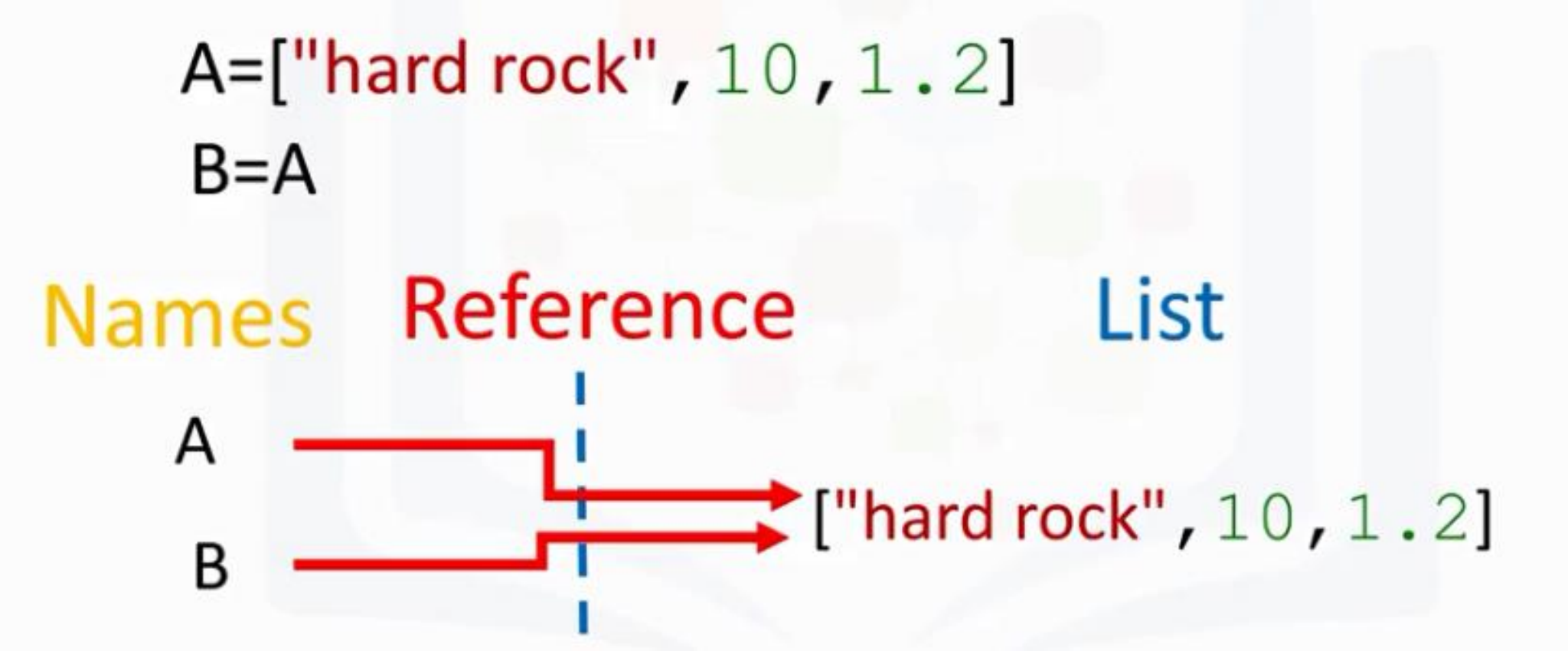
- What is the result of the following: "Hello Mike".split()
- ["Hello","Mike"]
- ["HelloMike"]
- ["H"]

Convert String to List

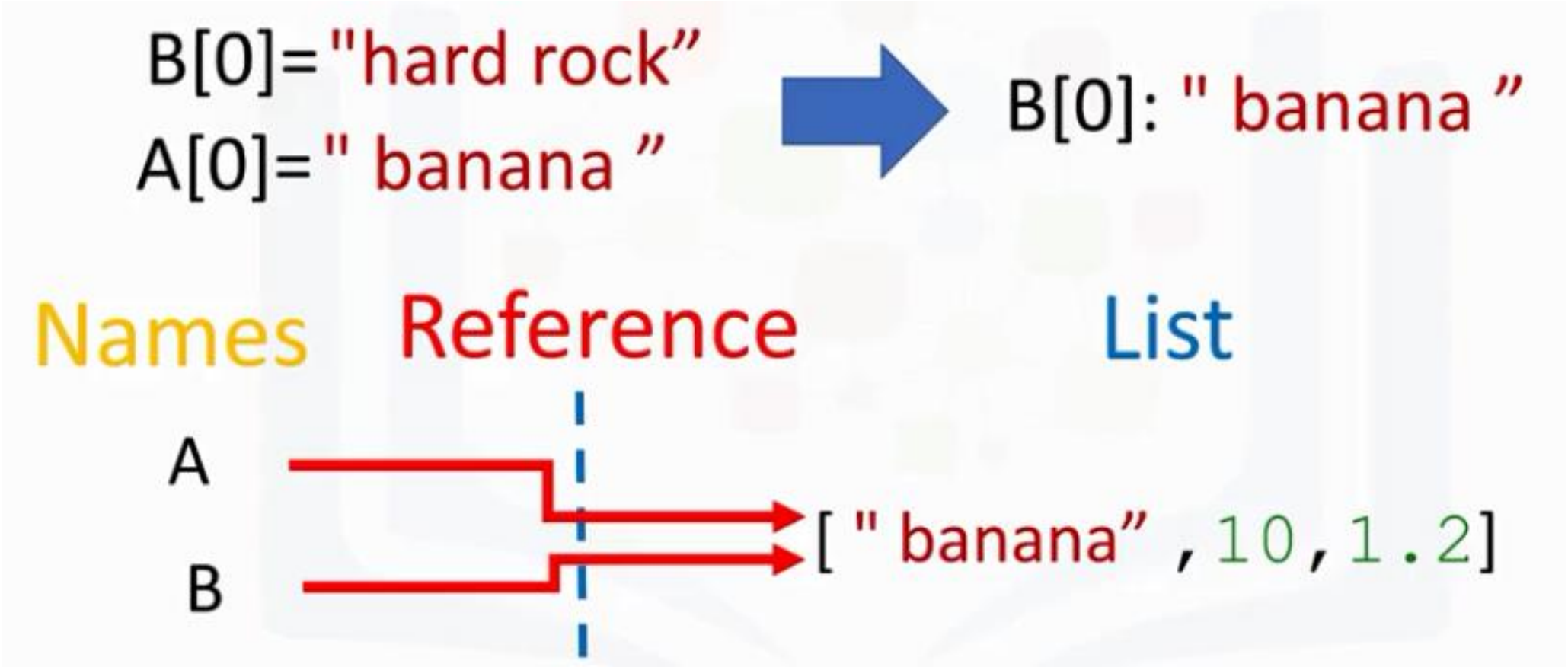
- We can use the split function to separate strings on a specific character known as a delimiter.



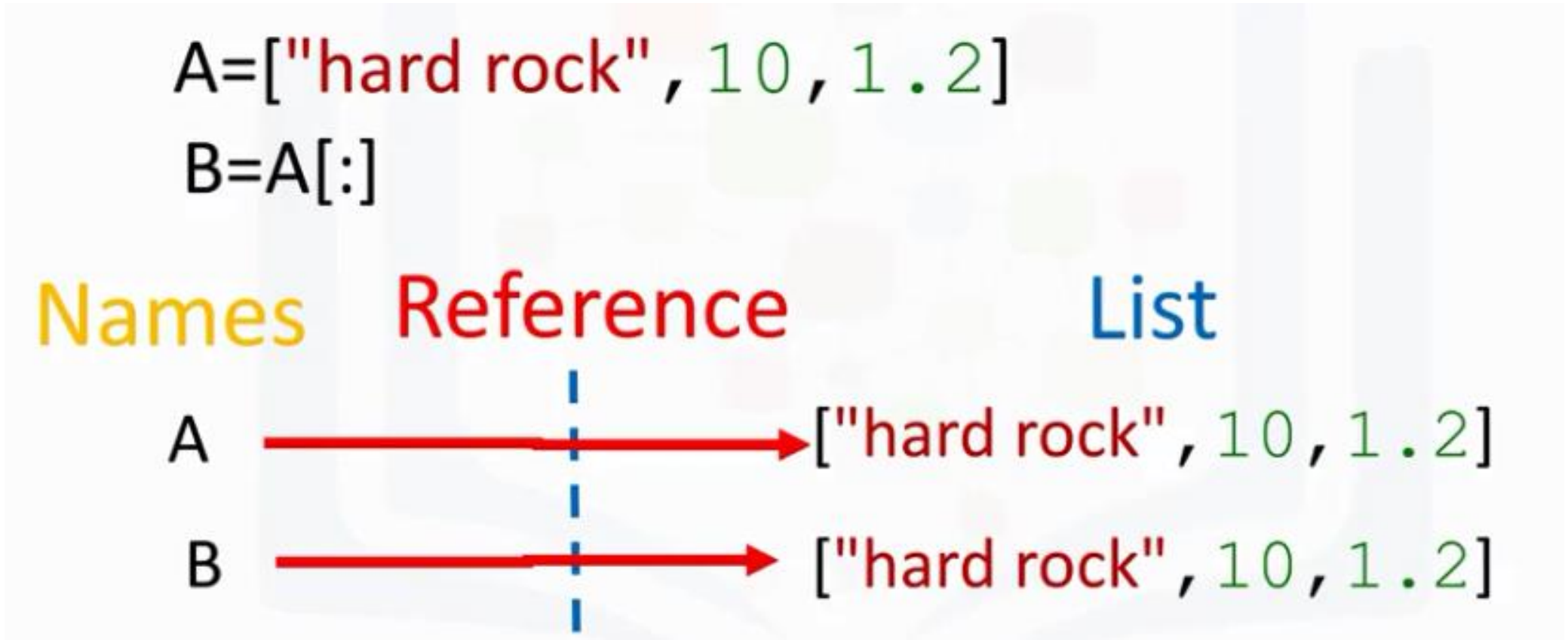
Lists: Aliasing



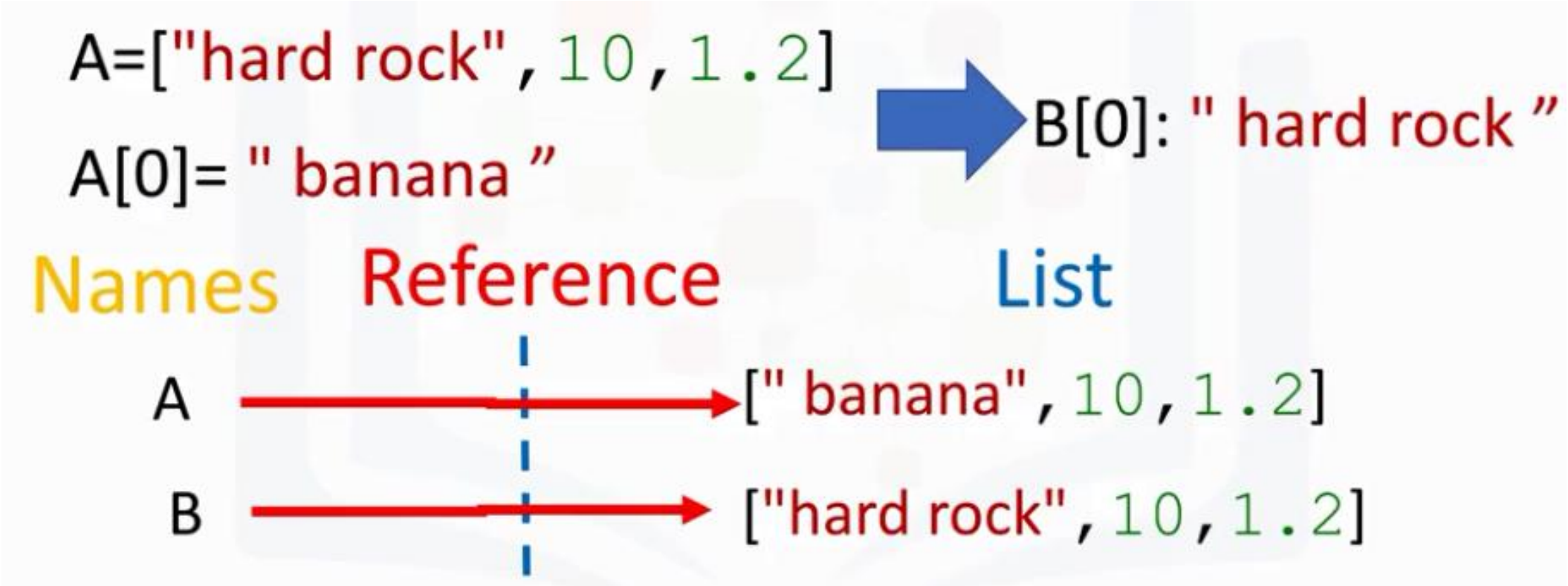
Lists: Aliasing



Lists: Clone


A=["hard rock",10,1.2]
B=A[:]
Names
Reference
List
A
["hard rock",10,1.2]
B
["hard rock",10,1.2]

Lists: Clone

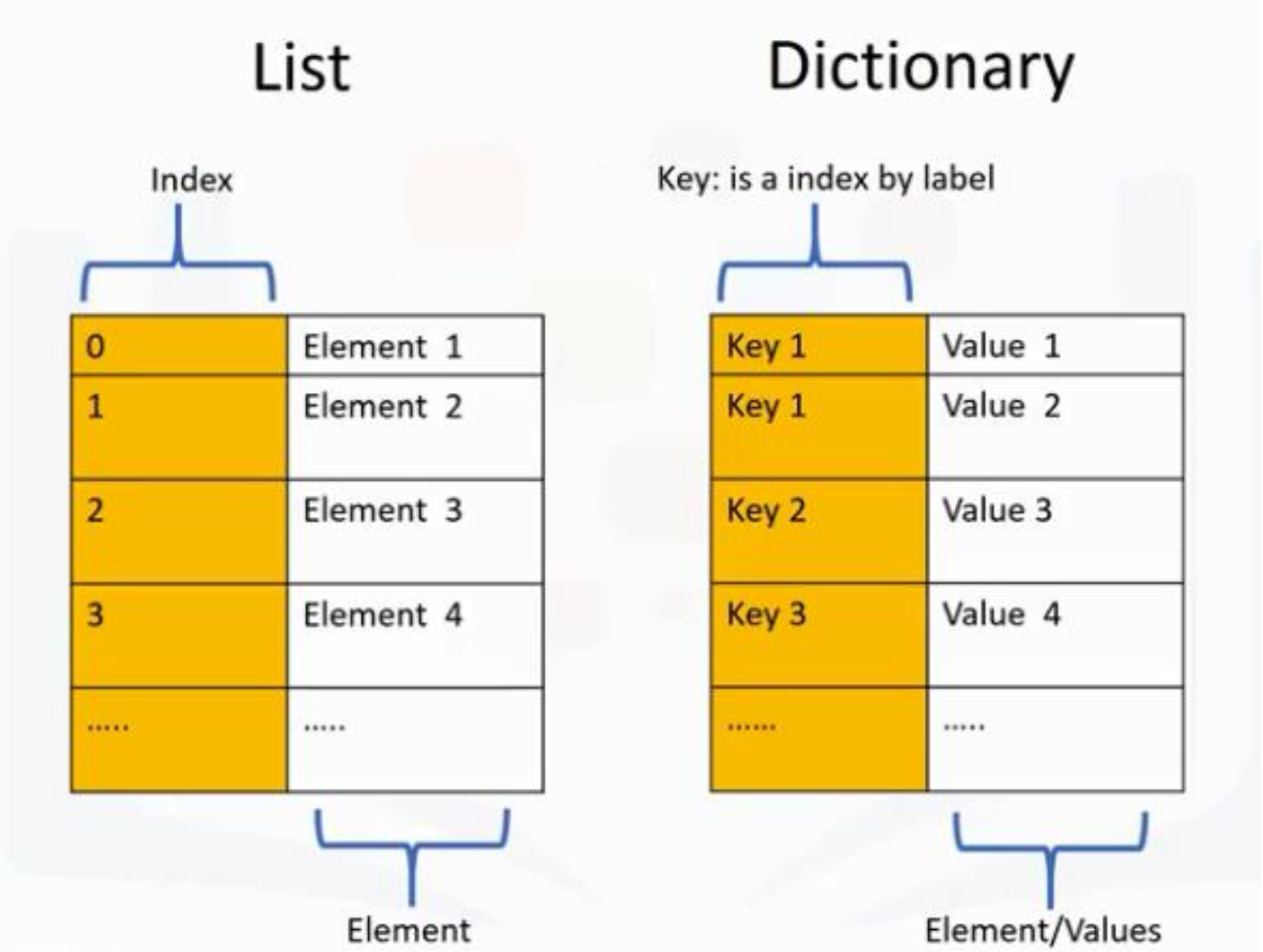

A=["hard rock",10,1.2]
A[0]= " banana "
B[0]: " hard rock "
Names
Reference
List
A
[" banana",10,1.2]
B
["hard rock",10,1.2]

Python Help

- We can get more info on list, tuples, and many other objects in Python using the help command.
- Simply pass in the list, tuple, or any other Python object.

```
A=["hard rock",10,1.2]

help(A)
```

Python Dictionaries

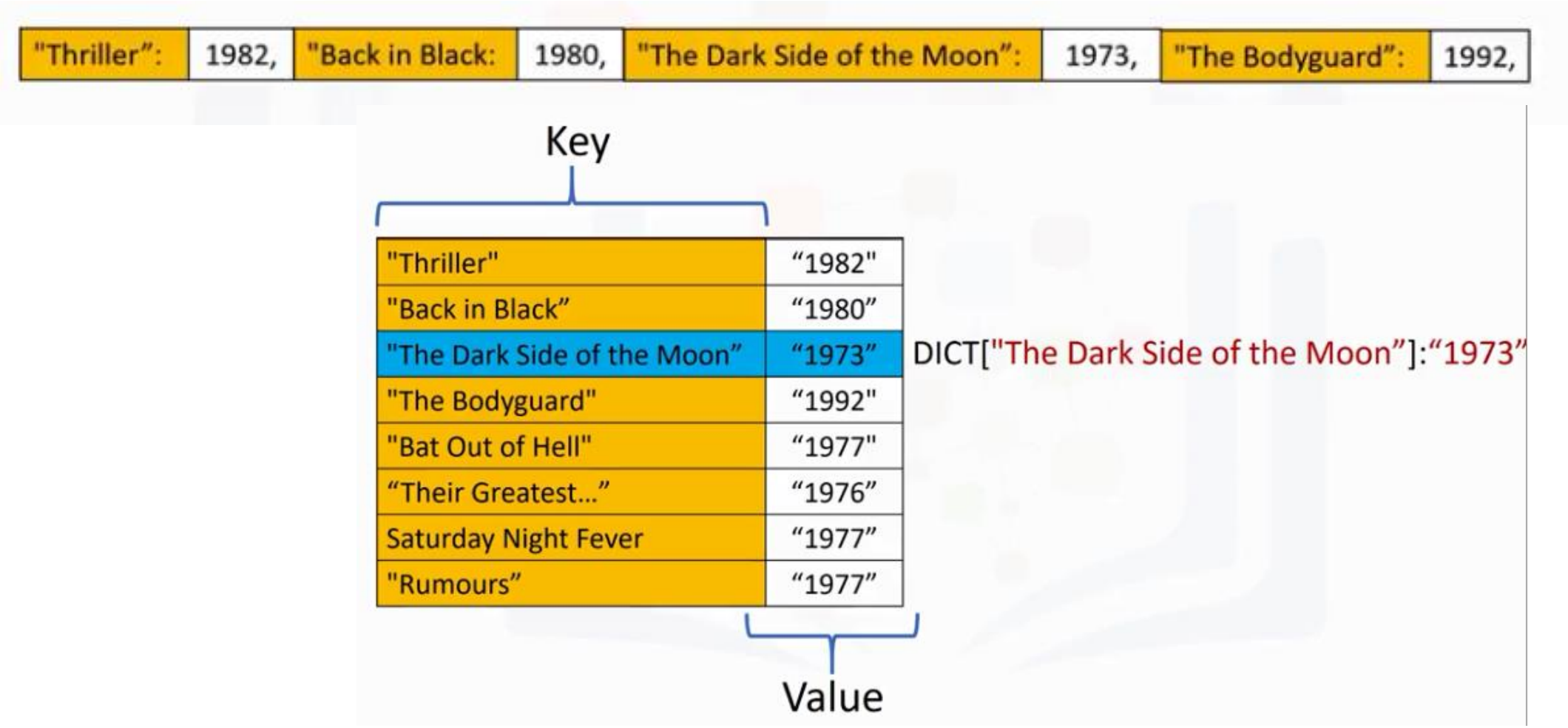


Dictionaries

- To create a dictionary, we use curly brackets {}.
- The keys are the first elements. They must be immutable and unique.
- Each key is followed by a value separated by a colon.
- The values can be immutable, mutable, and duplicates.
- Each key and value pair is separated by a comma.

```
{“key1”:1, "key2 ” :"2","key3” :[3,3,3], "key4":(4,4,4), ('key5’):5}
```

Dictionaries



Dictionaries

- Consider the following Python Dictionary:

  Dict={"A":1,"B":"2","C":[3,3,3],"D":(4,4,4),'E':5,'F':6}

- What is the result of the following operation: **Dict["D"]**

Add a new entry

"Thriller"	“1982"
"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
Saturday Night Fever	“1977”
"Rumors”	“1977”
'Graduation'	“2007”

```
DICT['Graduation']='2007'
```

Delete an entry

"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
Saturday Night Fever	“1977”
"Rumors”	“1977”

```
del(DICT['Thriller'])
```

To find from the dictionary

"Thriller"	“1982"
"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
Saturday Night Fever	“1977”
"Rumors”	“1977”

'The Bodyguard' in DICT

True

Get Keys

"Thriller"	“1982"
"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
“Saturday Night Fever”	“1977”
"Rumors”	“1977”

DICT.keys()=["Thriller”, "Back in Black”, "The Dark Side of the Moon”, "The Bodyguard”, "Bat Out of Hell”, “Their Greatest…”,“Saturday Night Fever”, "Rumors”]

Get Values

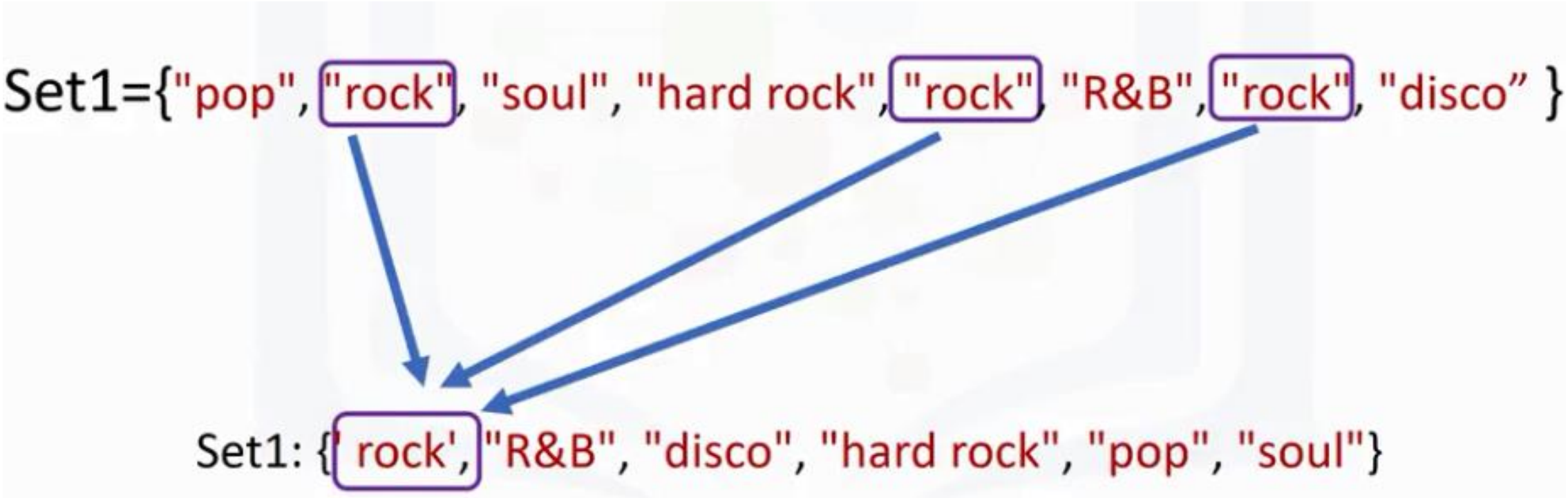
"Thriller"	“1982"
"Back in Black”	“1980”
"The Dark Side of the Moon”	“1973”
"The Bodyguard"	“1992"
"Bat Out of Hell"	“1977"
“Their Greatest…”	“1976”
Saturday Night Fever	“1977”
"Rumors”	“1977”

DICT.values() =[“1982”,“1980”,”1973”,“1992”, “1977”,“1976” “1977”, “1977”]

Sets

- Sets are types of collection
  - This means that like lists and tuples you can input different Python types
- Unlike lists and tuples they are unordered
  - This means sets do not record element position
- Sets only have unique elements
  - This means there is only one of a particular element in a set

Sets: Creating a Set



Sets: Creating a Set

- Consider the following set: **{"A","A"}**, what will the result be when the set is created

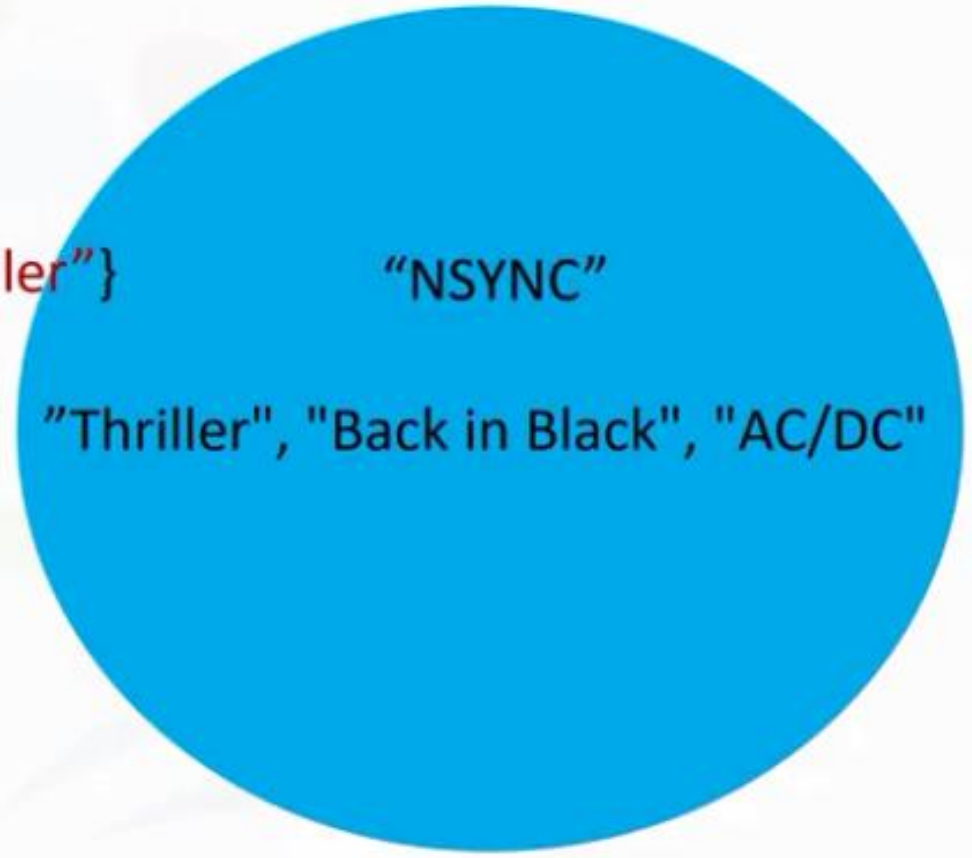
Sets: Creating a Set

Set Operations: Add element

A = {“Thriller”, “Back in Black”, “AC/DC”}

A.add("NSYNC")

A:{“AC/DC”, “Back in Black”, “NSYNC”, “Thriller”}

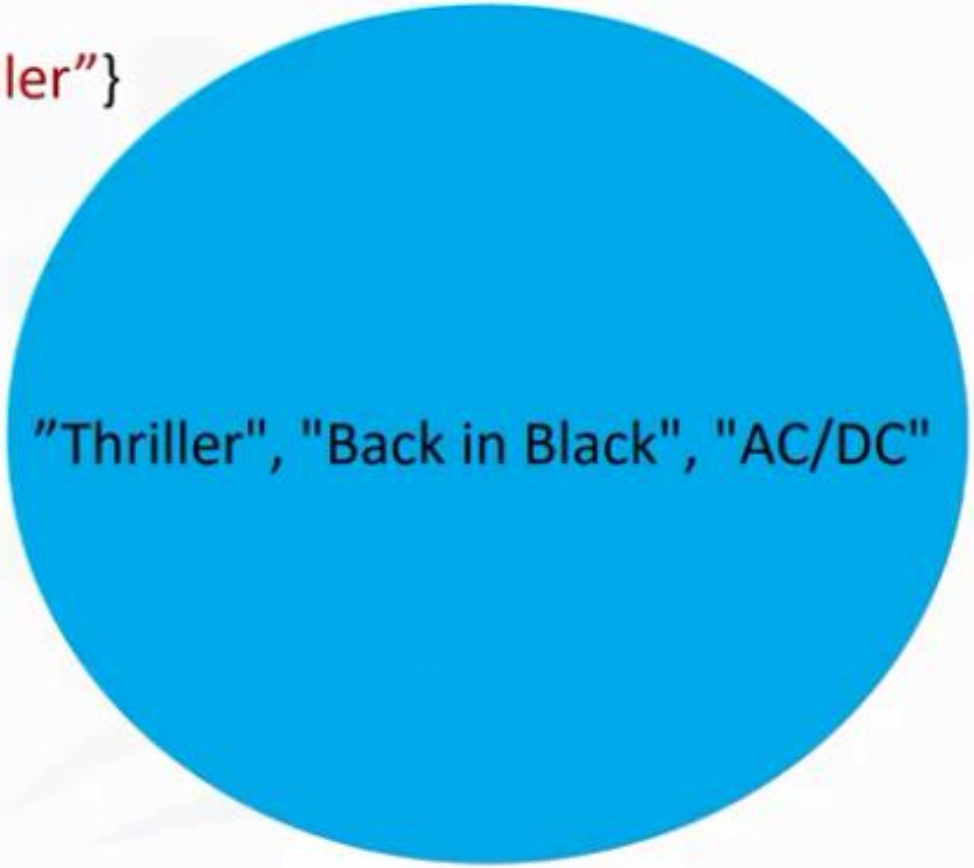


Set Operations: Remove element

A :{“AC/DC”, “Back in Black”, “NSYNC”, “Thriller”}

A.remove("NSYNC")

A:{“AC/DC”, “Back in Black”, “Thriller”}

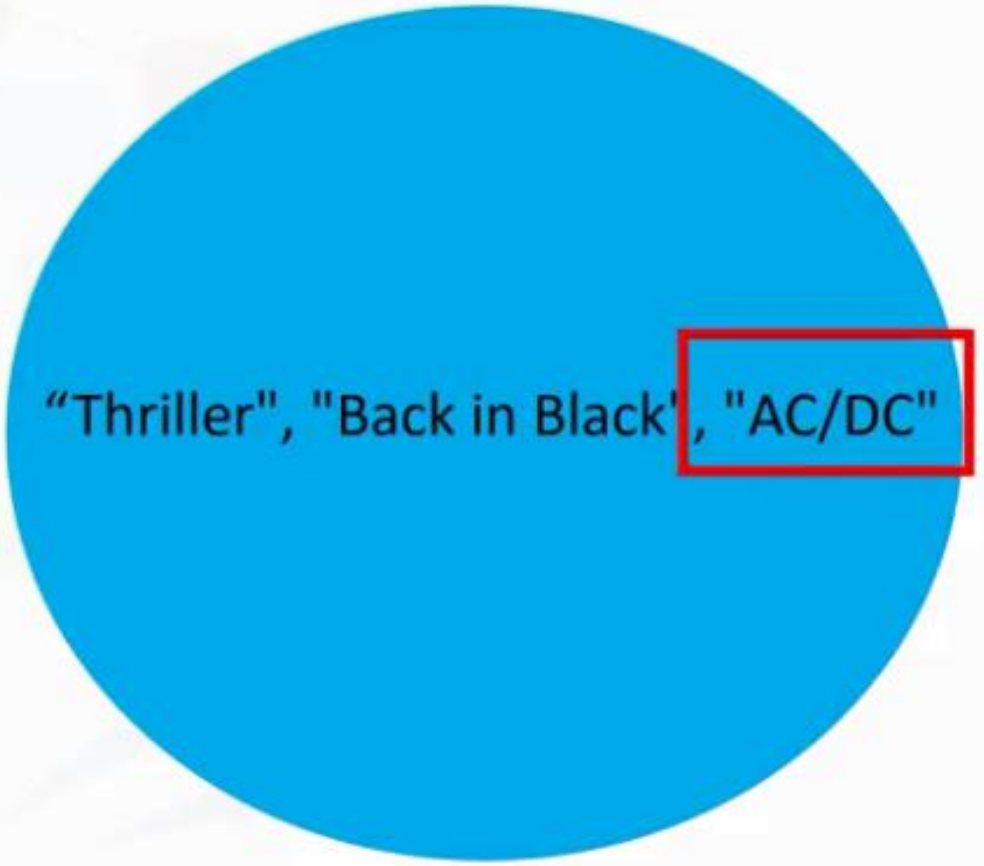


Set Operations: Find element

A:{“AC/DC”, “Back in Black”, “Thriller”}

“AC/DC” **in** A

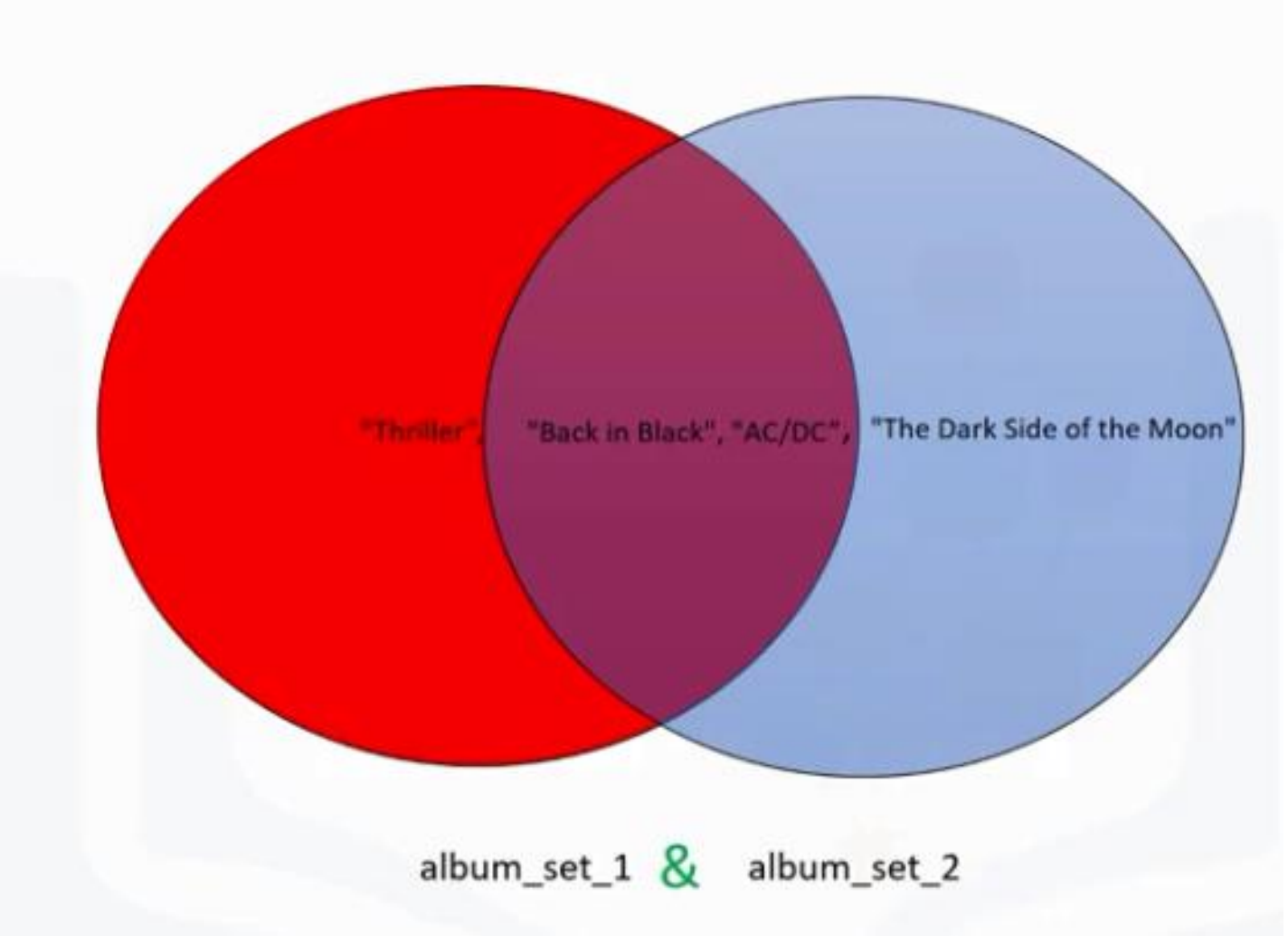
True



Sets: Mathematical set operations (Intersection)

```
album_set_1 ={"AC/DC"," Back in Black ", "Thriller }
album_set_2={"AC/DC", "Back in Black", "The Dark Side of the Moon"}
album_set_3=album_set_1 & album_set_2
album_set_3: {"AC/DC"," Back in Black "}
```

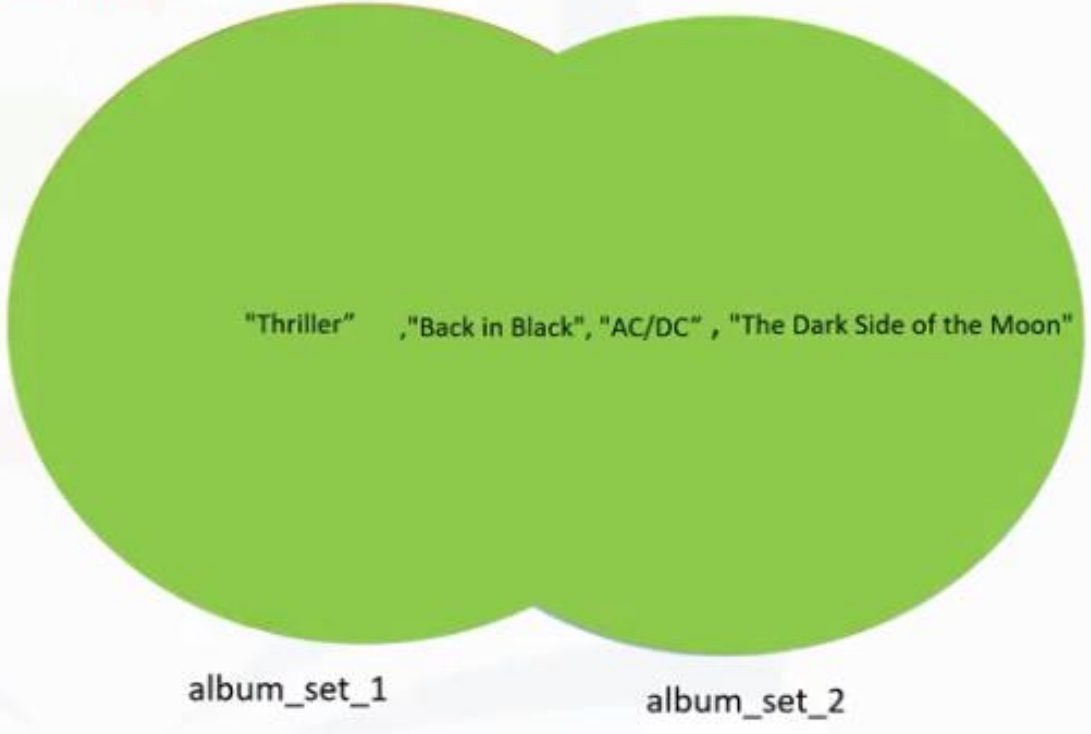
Sets: Mathematical set operations (Intersection)



Sets: Mathematical set operations (Union)

album_set_1. union(album_set_2)

{AC/DC", "Back in Black", "The Dark Side of the Moon", “Thriller” }



Sets: Mathematical set operations (Subset)