15-110 Principles of Computing – F21

Lecture 12:
Lists 2

Guest Teacher:
Hend Gedawy

Teacher:
Gianni A. Di Caro

Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements
- Removing List Elements
- Counting Element Occurrences in Lists/Tuples
- Finding Element Position in Lists/Tuples
- Comparing Lists/Tuples
- Finding Min/Max Elements in Lists/Tuples
- Summing Elements of Lists/Tuples
- Reversing Lists
- Sorting Lists

Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements
- Removing List Elements
- Counting Element Occurrences in Lists/Tuples
- Finding Element Position in Lists/Tuples
- Comparing Lists/Tuples
- Finding Min/Max Elements in Lists/Tuples
- Summing Elements of Lists/Tuples
- Reversing Lists
- Sorting Lists

Tuples vs. Lists

- Lists: [ ]
- Tuples: ( )

```
L = [3, 5, 7, 11]
L = (3, 5, 7, 11)
```

~ similar but

- Lists are **mutable** objects: can be changed!
- Tuples are **immutable** objects: cannot be changed!

```
L = [3, 5, 7, 11]
L[2] = -1
```

→ **`L = [3, 5, -1, 11]`**

```
T = (3, 5, 7, 11)
T[2] = -1
```

→ ERROR!

Tuples vs. Lists

- Lists are **mutable** objects: can be changed!
- Tuples are **immutable** objects: cannot be changed!

Reminder- Slicing


```
L = [3, 5, 7, 11]
xl = L[1:3]
xl[1] = 0
```

→ `xl = [5, 0]`

```
T = (3, 5, 7, 11)
xt = T[1:3]
xt[1] = 0
```

Slicing Ok → `xt` is a tuple!

ERROR!

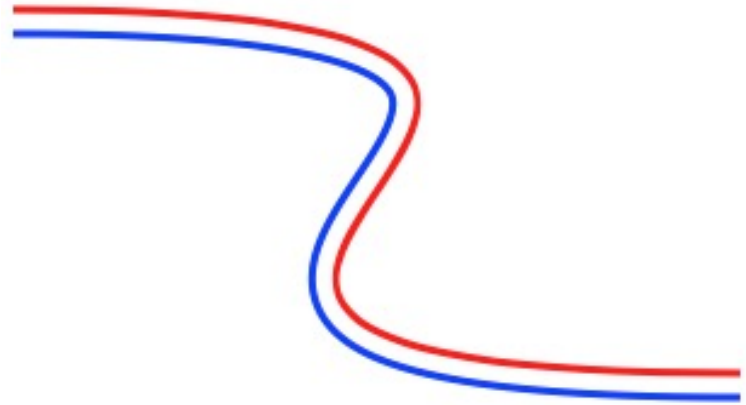
TypeError: 'tuple' object does not support item assignment

Why to use tuples? → To ensure / represent that a list of values won't be changed!

A tuple is a ***constant/fixed*** **list!**

Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements
- Removing List Elements
- Counting Element Occurrences in Lists/Tuples
- Finding Element Position in Lists/Tuples
- Comparing Lists/Tuples
- Finding Min/Max Elements in Lists/Tuples
- Summing Elements of Lists/Tuples
- Reversing Lists
- Sorting Lists



Parallel assignments

```
T = (4,3,2,1)

a = T[0]
b = T[1]
c = T[2]
d = T[3]

print("a =", a)
print("b =", b)
print("c =", c)
print("d =", d)
```

A more compact way of making the same assignment:

```
T = (4,3,2,1)
a, b, c, d = T

print("a =", a)
print("b =", b)
print("c =", c)
print("d =", d)
```

- **Number of values on the right must be the same as the number of variables on the left**

```
L = [1,2,3]

a, b = L
```

ValueError: too many values to unpack (expected 2)

Lists/Tuples Operations

Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements
- Removing List Elements
- Counting Element Occurrences in Lists/Tuples
- Finding Element Position in Lists/Tuples
- Comparing Lists/Tuples
- Finding Min/Max Elements in Lists/Tuples
- Summing Elements of Lists/Tuples
- Reversing Lists
- Sorting Lists

Extending a list by adding multiple list elements: +

- **Concatenation** operator **+** : add items from another list onto the **end** of the list

```
primes = [3, 5, 7, 11, 13, 17]
primes = primes + [19, 23, 29]
```

→ primes it's a *new* list with **[3,5,7,11,13,17,19, 23, 29]**

```
prime_numbers = [3, 5, 7, 11]
other_primes = [13, 17, 19]
new_primes = prime_numbers + other_primes
```

→ *new* list with **[3,5,7,11,13,17,19]**

Adding list elements: + operator

- The **+** operator concatenates two lists and creates a **NEW one**

```
primes = [2, 3, 5, 7, 11, 13]
primes2 = [17, 19, 23]
primes = primes + primes2
```

```
primes ?
→ [2, 3, 5, 7, 11, 13, 17, 19, 23]
```

➢ **Is `primes` the *same* list as before?** i.e., is primes at the same place in the memory?

No: a new list is created and stored in some (other) memory address → **Expensive!**

```
primes = [2, 3, 5, 7, 11, 13]
print('Original address of primes:', id(primes))
primes2 = [17, 19, 23]
primes = primes + primes2
print('New address of primes:', id(primes))
```

Adding single list elements: + operator

- We can use the **+** operator to add one single element to the list (need to use [ ])

```
primes = [2, 3, 5, 7, 11, 13]
primes = primes + [17]
```

```
primes ?
→ [2, 3, 5, 7, 11, 13, 17]
```

➢ **Remember: after this operation a new list is being created in memory**

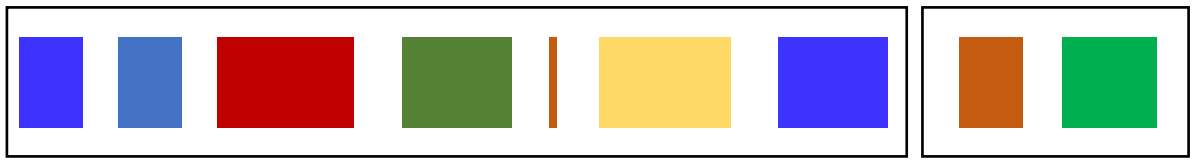
```
primes = [1, 3, 5, 7, 11, 13, 17]
primes = primes + 19
```

→ ERROR!

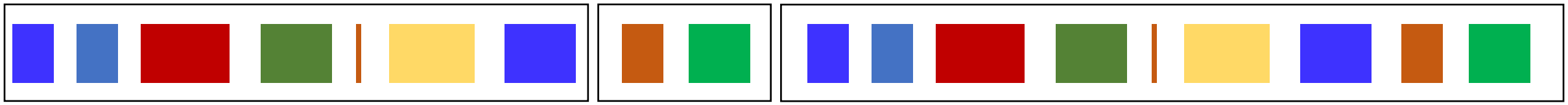
Extending a list by adding multiple list elements: +

- The + operator concatenates two lists and creates a **NEW one**

Memory BEFORE (+) Operation

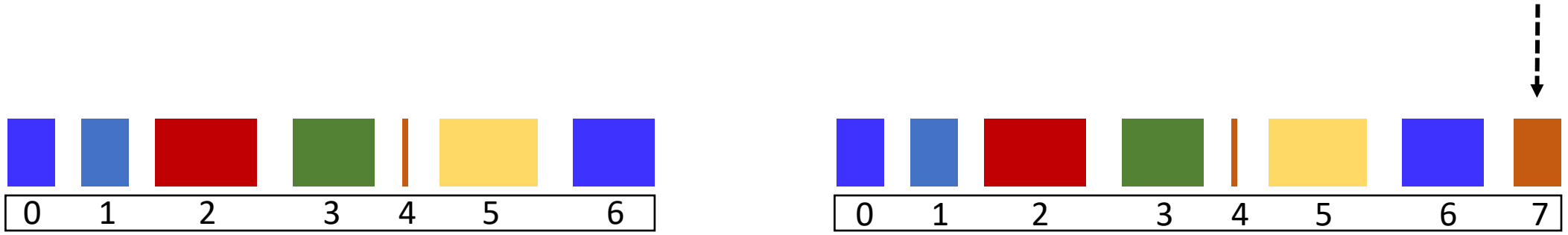


Memory AFTER (+) Operation



Adding single list elements: `.append()` method

- Method `L.append(item)`: add an item at the end of the **same list** (*in-place*)



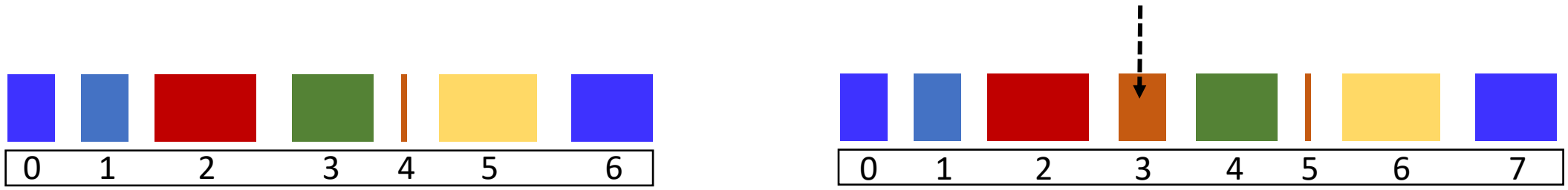

```
primes = [2, 3, 5, 7, 11, 13, 17]
```

`primes.append(19)` → same list, extended to the end by adding one `int` literal of value 19

```
primes → [2, 3, 5, 7, 11, 13, 17, 19]
```

Adding single list elements: `.insert()` method

- Method: `L.insert(index, item)`: add an item at the `index` position of the **same list** (*in place*), moving all the other items in the list up by one index number

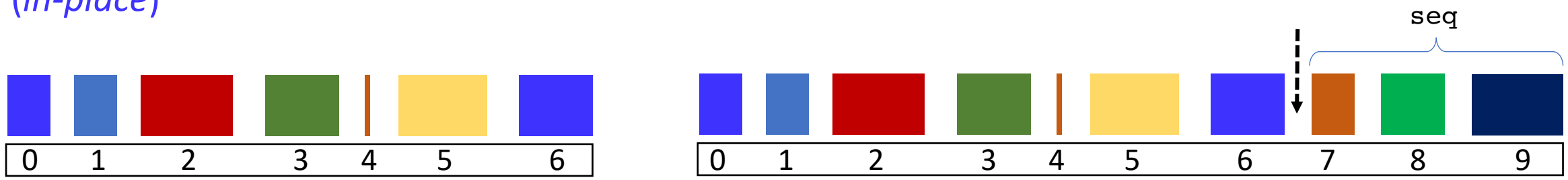



```
primes = [2, 3, 5, 7, 11, 13, 17]
primes.insert(3,19)
```
→ same list, with new item: [2, 3, 5, 19, 7, 11, 13, 17]

```
primes = [2, 3, 5, 7, 11, 13, 17]
primes.insert(0,19)
```
→ same list, with new item, all positions shifted: [19, 2, 3, 5, 7, 11, 13, 17]

Adding multiple list elements: `.extend( )` method

- Method `L.extend(seq)`: add all items from another **list/tuple** onto the end of the **same list** `L` (*in-place*)




```
primes = [2, 3, 5, 7, 11, 13, 17]
other_primes = (19, 23, 29)
```

`primes.extend(other_primes)` → same list, extended at the end by adding `other_primes`

`primes` →[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

`primes.extend(other_primes[0:2])` → extended at the end by adding two items of `other_primes`

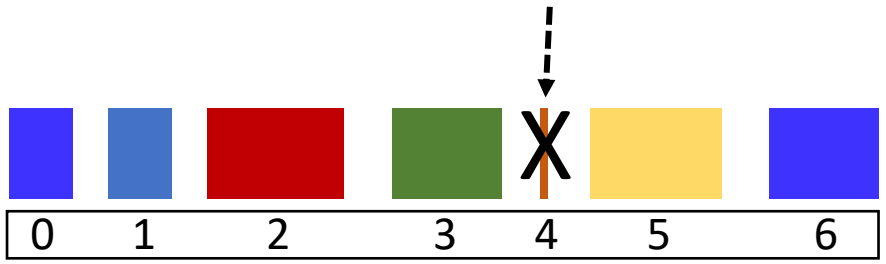
`primes` →**[2, 3, 5, 7, 11, 13, 17, 19, 23]**

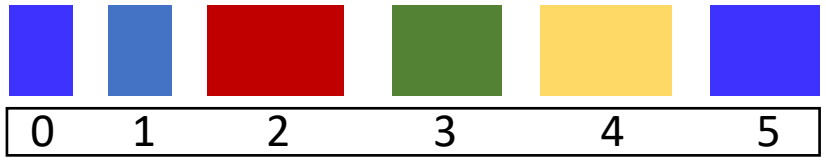
Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding Elements: +, L.append(), L.insert(), L.extend()
- Removing List Elements
- Counting Element Occurrences in Lists/Tuples
- Finding Element Position in Lists/Tuples
- Comparing Lists/Tuples
- Finding Min/Max Elements in Lists/Tuples
- Summing Elements of Lists/Tuples
- Reversing Lists
- Sorting Lists

Removing single list elements: `.remove()` method

- Method `L.remove(item)`: remove the (first) element with value `item` in the list, moving all the other items in the list down by one index number (*in-place*)
- → Removal **by content**





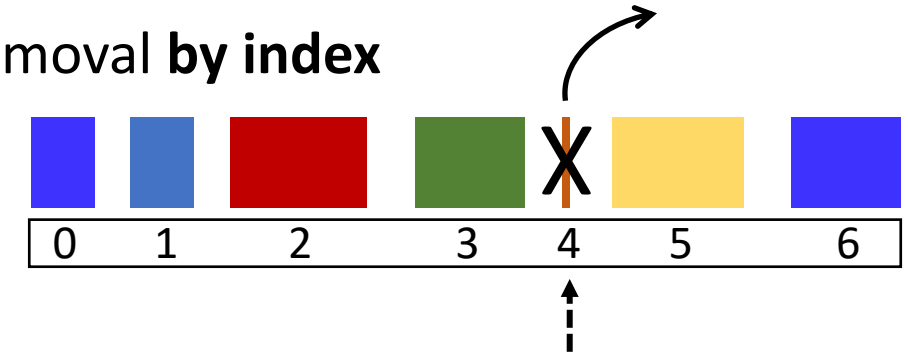

```
numbers = [1, 3, 5, 4, 5, 5, 17]
```

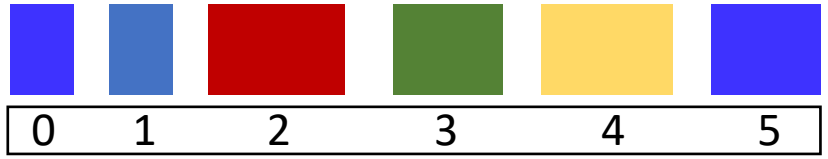
`numbers.remove(5)` → same list, with the first element of value 5 being removed: **[1,3,4,5,5,17]**

`numbers.remove(15)` → ERROR! an item with value 15 is not found in the list

Removing single list elements: `.pop()` method

- Method `L.pop(index)`: takes the argument `index` and removes the item present at that index, moving all the other items in the list up by one index number (*in-place*)
- The removed item is also **returned** by the function
- → Removal **by index**





`numbers = [1, 3, 5, 4, 5, 5, 17]`

`numbers.pop(2)` → same list, with the item at index 2, of value 5, being removed: **[1,3,4,5,5,17]**

`numbers.pop()` → same list, with last item removed: **[1,3,5,4,5,5]**

`n = numbers.pop(0)` → n gets value **1**

`numbers.pop(8)` → ERROR! an index 8 is out of range for the list: **need to use `len()` prior to `pop()`**

```
if 8 < len(numbers):
    numbers.pop(8)
```

Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements: +, L.append(), L.insert(), L.extend()
- Removing List Elements: L.remove() , L.pop()
- Counting Element Occurrences in Lists/Tuples
- Finding Element Position in Lists/Tuples
- Comparing Lists/Tuples
- Finding Min/Max Elements in Lists/Tuples
- Summing Elements of Lists/Tuples
- Reversing Lists
- Sorting Lists

Count how many occurrences of an item: `.count()` method

- `L.count(item)` : Returns the number of occurrences of `item` in the **list/tuple** `L`

  - **List:**

    ```
    scores = [1, 11, 5, 11, 4, 11, 7, 9, 0, 4]
    n = scores.count(11)
    ```

    → `n` is an integer of value **3**, the # of occurrences of 11 in `scores`

  - **Tuple:**

    ```
    l1 = (True, False, True).count(True)
    ```

    → `l1` is an integer of value **2** (two occurrences of `True`)

    ```
    l2 = (True, False, True).count(1)
    ```

    → `l2` is an integer of value **2** (two occurrences of `True`)

Reminder
True is equivalent to 1
False is equivalent to 0

Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements: +, L.append(), L.insert(), L.extend()
- Removing List Elements: L.remove() , L.pop()
- Counting Element Occurrences in Lists/Tuples: L.count()
- Finding Element Position in Lists/Tuples
- Comparing Lists/Tuples
- Finding Min/Max Elements in Lists/Tuples
- Summing Elements of Lists/Tuples
- Reversing Lists
- Sorting Lists

Get the position of an item: `.index()` method

- `L.index(item)` : Returns the index of the first occurrence of `item` in the **list/tuple** `L`

  - **List**

```
scores = [1, 11, 5, 11, 4, 11, 7, 9, 0, 4]
```

`ns = scores.index(11)` → `ns` is an integer of value **1**, the index of first occurrence of 11 in `scores`

`ns = scores.index(19)` → generates an ERROR since 19 is not in `scores`: **to avoid the error use the operator `in` to check membership first**

```
if 19 in scores:
    ns= scores.index(19)
```

  - **Tuple**

```
T = (True, False, True)
```

`nt= T.index(True)` → `nt` is an integer of value **0**, the index of first occurrence of True

`nt= T.index(0)` → `nt` is an integer of value **1**, the index of first occurrence of 0

Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements: +, L.append(), L.insert(), L.extend()
- Removing List Elements: L.remove() , L.pop()
- Counting Element Occurrences in Lists/Tuples: L.count()
- Finding Element Position in Lists/Tuples: L.index()
- Comparing Lists/Tuples
- Finding Min/Max Elements in Lists/Tuples
- Summing Elements of Lists/Tuples
- Reversing Lists
- Sorting Lists

Comparison between lists / tuples: <, >, >=, <=, ==, !=

✓ **Comparison operators** can be applied to list/tuples!

- <
- >
- >=
- <=
- ==
- !=

```
L1 = [0, 1, 5, -5]
L2 = [1, 3, 6, 0]
L3 = [0, 1, 4, 7]
L4 = [1, 3, 6, 0]

L1 > L2 ?   → False
L1 == L3 ?  → False
L1 > L3 ?   → True
L2 > L3 ?   → True
L4 == L2 ?  → True
```

Comparison between two lists L1, L2, happens in *lexicographic order:*

1. Compare the **first element**:
   - if L1[0] > L2[0] → L1 > L2
   - elif L2[0] > L1[0] → L2 > L1
   - else (L1[0] is the same as L2[0]):
     2. compare the **second element**:
        - if L1[1] > L2[1] → L1 > L2
        - elif L2[1] > L1[1] → L2 > L1
        - else (L1[1] is the same as L2[1]):
          3. compare the **third element**:
             - if L1[2] > L2[2] → L1 > L2
             - elif L2[2] > L1[2] → L2 > L1
             - else (L1[2] is the same as L2[2]):
               4. …

Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements: +, L.append(), L.insert(), L.extend()
- Removing List Elements: L.remove() , L.pop()
- Counting Element Occurrences in Lists/Tuples: L.count()
- Finding Element Position in Lists/Tuples: L.index()
- Comparing Lists/Tuples: <, > , =, !=, <=, >=, ==
- Finding Min/Max Elements in Lists/Tuples
- Summing Elements of Lists/Tuples
- Reversing Lists
- Sorting Lists

Finding minimum and maximum: `min()`, `max()` functions

- `min(L)` : Returns the **item** of the **list/tuple** `L` with the **minimum value**
- `max(L)` : Returns the **item** of the **list/tuple** `L` with the **maximum value**
  - → Return type depends on the type of the items
  - Without a key (optional argument for comparison), it can be applied only to homogeneous lists/tuples (all elements of the same type)

List:

```
prime_numbers = [2, 3, 5, 7, 11]
n = max(prime_numbers)
n = min(prime_numbers)
```

→ `n` is an integer of value 11, the item of highest value

→ `n` is an integer of value 2, the item of lowest value

Tuple:

```
logical = max(True, False, True)
```

→ `logical` is a boolean of value `True` (1)

```
x = max(1, 3, True, 'red')
x = min([1, 2, 3, [7,8]])
```

→ generates an **ERROR** (how to compare different items?)

Lecture Outline - Lists Operations

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements: +, L.append(), L.insert(), L.extend()
- Removing List Elements: L.remove() , L.pop()
- Counting Element Occurrences in Lists/Tuples: L.count()
- Finding Element Position in Lists/Tuples: L.index()
- Comparing Lists/Tuples: <, > , =, !=, <=, >=, ==
- Finding Min/Max Elements in Lists/Tuples: min(L), max(L)
- Summing Elements of Lists/Tuples
- Reversing Lists
- Sorting Lists

Summing up all the elements in the list/tuple: `sum( )` function

- `sum(L)` : Returns the **sum** of the elements in the **list/tuple** `L`

List:

```
numbers = [1, 2, 3, 4, 5]
```

`n = sum(numbers)` → `n` is an integer of value **15**, the sum of the 5 items

```
mix = [1, 2.5, 3, 4.6, 5]
```

`n = sum(mix)` → `n` is a float of value **16.1**, the sum of the 5 items

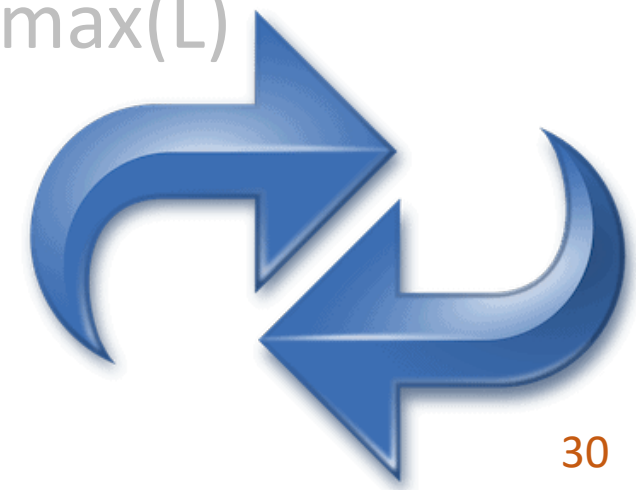
Tuple:

```
logical = (True, False, True)
```

`n = sum(logical)` → `n` is an integer of value **2**

Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements: +, L.append(), L.insert(), L.extend()
- Removing List Elements: L.remove() , L.pop()
- Counting Element Occurrences in Lists/Tuples: L.count()
- Finding Element Position in Lists/Tuples: L.index()
- Comparing Lists/Tuples: <, > , =, !=, <=, >=, ==
- Finding Min/Max Elements in Lists/Tuples: min(L), max(L)
- Summing Elements of Lists/Tuples: sum(L)
- Reversing Lists
- Sorting Lists



Reverse the list : `.reverse()` and Slicing method

- `L.reverse()` : Changes (*in-place*) the list `L` (not applicable to tuples!) putting the elements in the reverse order compared to the original list

```
numbers = [1, 4, 2, -7, 0, 6]
numbers.reverse()
```

→ `numbers` list is now: **[6, 0, -7, 2, 4, 1]**

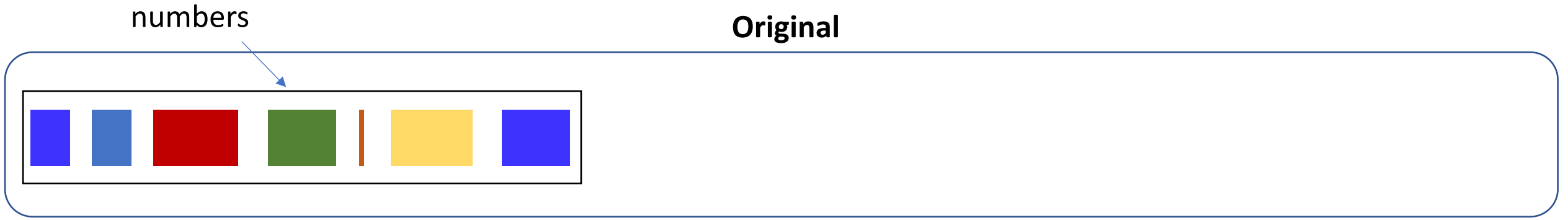
- Other way to obtain the same macroscopic result using `[ ]` operator with *slicing*:
  using `[::-1]`, you obtain a **copy** of the list `L` reversed

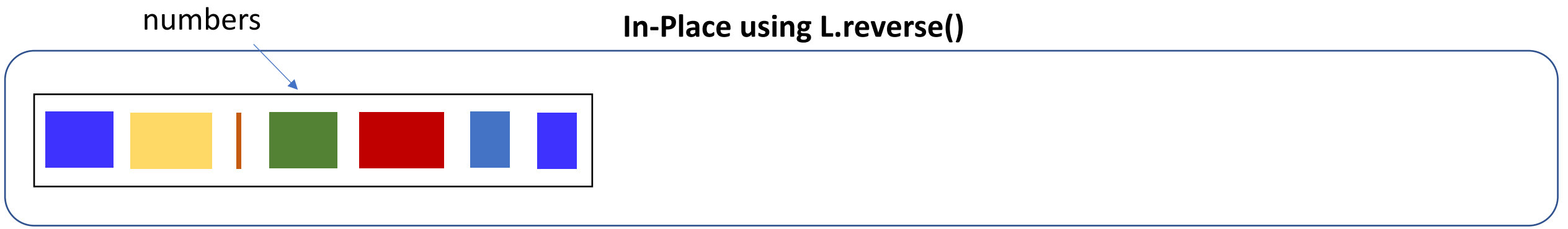
```
numbers = [1, 4, 2, -7, 0, 6]
r = numbers[::-1]
```

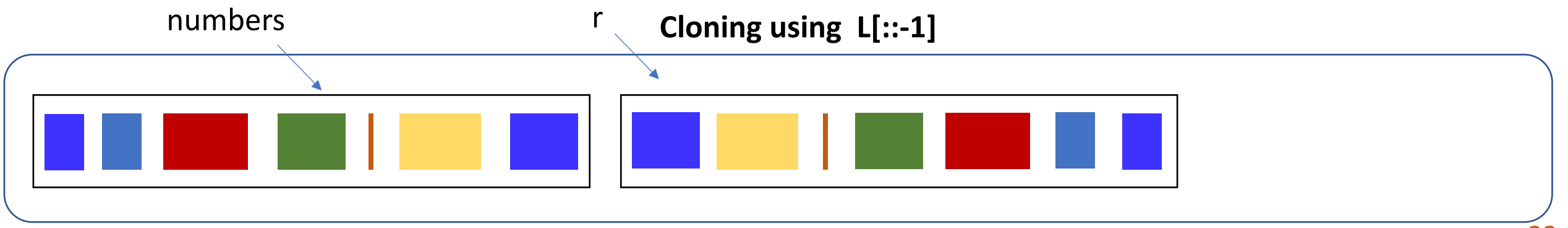
→ `r` is the list **[6, 0, -7, 2, 4, 1]**
→ `numbers` hasn't changed!
→ `r` and `numbers` have different identities

- **Watch out**:
  - In this case a list with a *new* identity is being created (but the *macroscopic* effect is the same)
  - In-place vs. cloning operations

Reversing – Cloning vs. In-Place

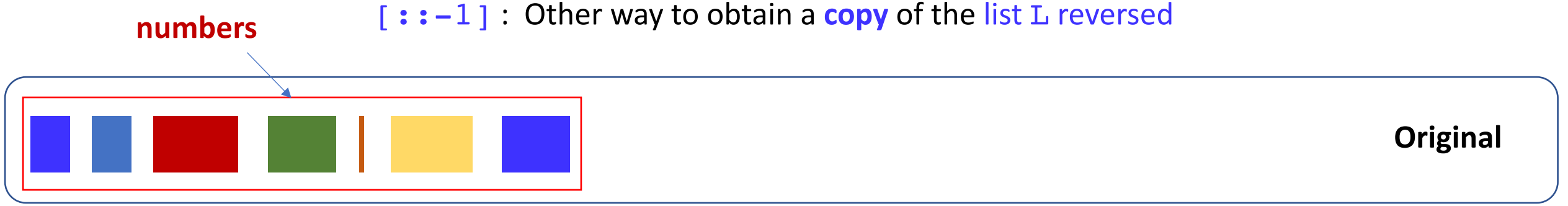






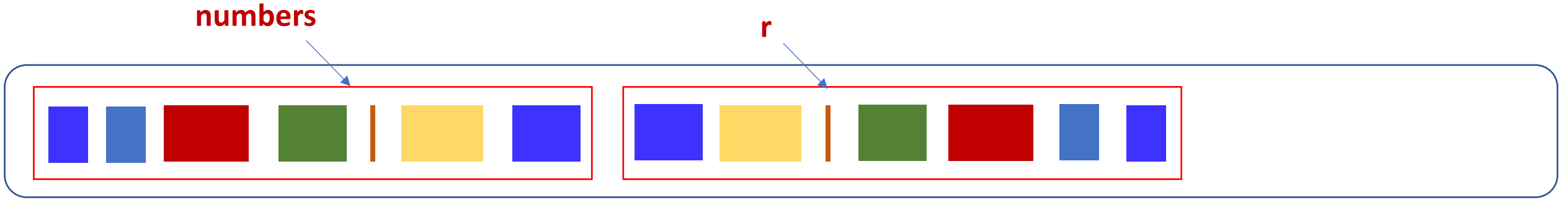
Get a reversed list: slicing (cloning)

`[::-1]` : Other way to obtain a **copy** of the list `L` reversed



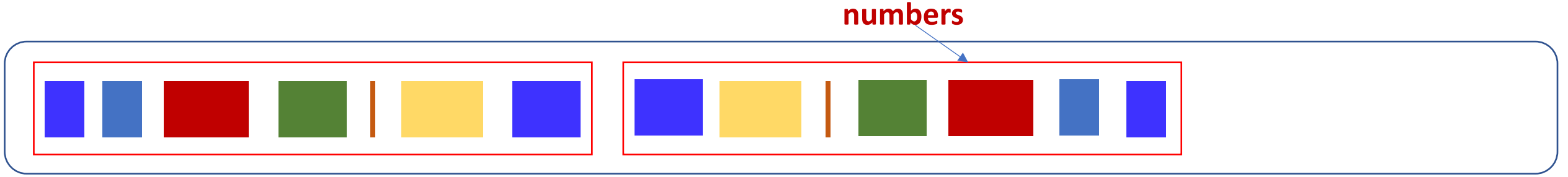

```
r = numbers[::-1]
```

→ `numbers` hasn't changed!
→ `r` and `numbers` have different identities




```
numbers = numbers[::-1]
```

→ `Original list` hasn't changed!
→ original list and `numbers` have different identities



Lecture Outline

- Lists vs Tuples
- Parallel Assignments
- Adding List Elements: +, L.append(), L.insert(), L.extend()
- Removing List Elements: L.remove() , L.pop()
- Counting Element Occurrences in Lists/Tuples: L.count()
- Finding Element Position in Lists/Tuples: L.index()
- Comparing Lists/Tuples: <, > , =, !=, <=, >=, ==
- Finding Min/Max Elements in Lists/Tuples: min(L), max(L)
- Summing Elements of Lists/Tuples: sum(L)
- Reversing Lists: L.reverse(), L[::-1]
- Sorting Lists

Get an ordered list/tuple: `sorted()` function

- `sorted(seq)`: works for any sequence (list, tuple, string) and returns a **list which is a sorted copy of the original sequence**
  - The original object is *not modified*

```
L = [2,4,1]

b = sorted(L)

print(L, b)            → [2, 4, 1] [1, 2, 4]
print(id(l), id(b))    →  4989597000 4584103560
```

`sorted()` *function* makes a copy of the object and returns it sorted

Get an ordered list/tuple: `sorted( )` function, reverse order

- By default, `sorted(seq)` orders the elements of `seq` in ascending order

```
L = [-1, 2, 7, 1, -2, 0, 5]

b = sorted(L)          → [-2, -1, 0, 1, 2, 5, 7]
```

- What about sorting in reverse, descending order?

- `sorted(seq, reverse=True)`, optional argument of the function

```
L = [-1, 2, 7, 1, -2, 0, 5]

b = sorted(L, reverse=True)  → [7, 5, 2, 1, 0, -1, -2]
```

Order a list in-place: `.sort()` method

- `L.sort()` : Changes (*in-place*) the list `L` (not applicable to tuples!) with the elements sorted in ascending order (by default)
  - The (optional) parameter `reverse`, if set to `True`, provides the result in *descending* order

```
L = [-1, 2, 7, 1, -2, 0, 5]
L.sort()
```

→ Now `L` is the list `[-2, -1, 0, 1, 2, 5, 7]`

```
L = [-1, 2, 7, 1, -2, 0, 5]
L.sort(reverse=True)
```

→ Now `L` is the list `[7, 5, 2, 1, 0, -1, -2]`

Sorting on list of lists / tuples: applies to both L.sort() and sorted(L)

- A *list of lists/tuples of primitive types* is sorted according to the **first element**(s) of each list/tuple

```
my_tuples = [(1,2), (5,7,8), (-1,), (0,9,1,3)]
my_tuples.sort()
```

→ `[(-1,), (0, 9, 1, 3), (1, 2), (5, 7, 8)]`

```
my_tuples = [(-1,2), (5,7,8), (-1,3), (0,9,1,3)]
m = sorted(my_tuples)
```

→ `[(-1,2), (-1,3), (0, 9, 1, 3), (5, 7, 8)]`

  - *Ties* do not matter since the items become indistinguishable

```
my_tuples = [(-1,2), (5,7,8), (-1,2), (0,9,1,3)]
my_tuples.sort()
```

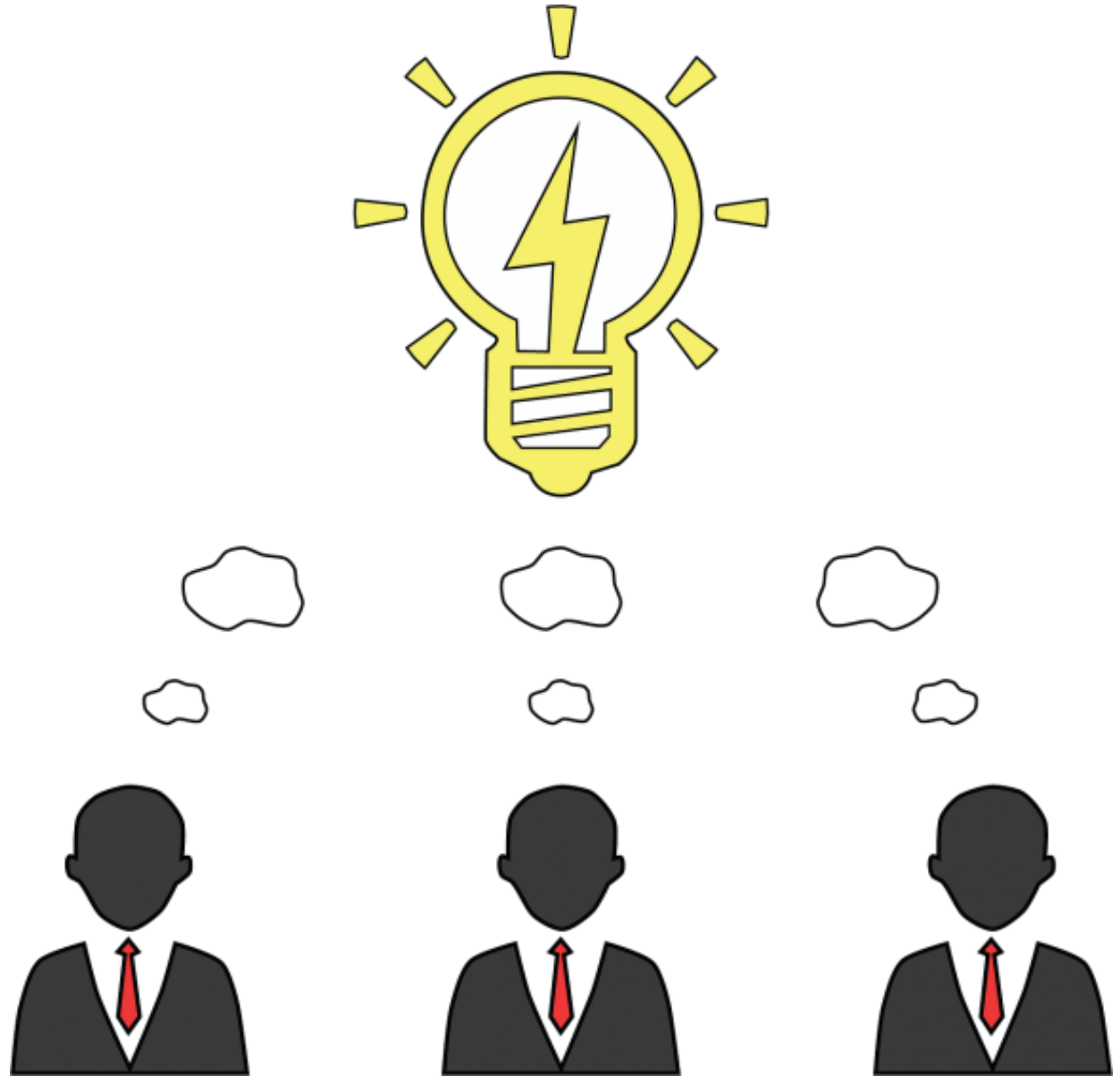
→ `[(-1,2), (-1,2), (0, 9, 1, 3), (5, 7, 8)]`


SUMMARY

Lists Operations Summary

- Adding List Elements: +, L.append(item), L.insert(index, item), L.extend(seq)
- Removing List Elements: L.remove(item) , L.pop(index)
- Counting Element Occurrences in Lists/Tuples: L.count()
- Finding Element Position in Lists/Tuples: L.index()
- Comparing Lists/Tuples: **<, > , =, !=, <=, >=, ==**
- Finding Min/Max Elements in Lists/Tuples: min(L), max(L)
- Summing Elements of Lists/Tuples: sum(L)
- Reversing Lists: L.reverse(), L[::-1]
- Sorting Lists: sorted(L), L.sort()

Test your knowledge



Test your knowledge (1)

What is the Output of the Following Code:

1)
```
names= ['Nora', 'Salem', 'Zeyad', 'Noof']
print(max(names)) → Zeyad
print(min(names)) → Noof
```

2)
```
even=[2,4,6,8,10]
even[0]=None
print(even) → [None, 4,6,8,10]
```

3)
```
odd=[1,3,5,7,9]
print(odd.pop()) → 9
print(odd.pop(3)) → 7
```

4)
```
T=(2,6,9,7,4,5,10)
print(sorted(T)) → [2,4,5,6,7,9,10]
```

5)
```
T1= ('a','d','b','g','e')
T2= ('c','f','h', 'i')
numbersL= [1,2,3,4,5]
print(T2+T1) → ('c', 'f', 'h', 'i', 'a', 'd', 'b', 'g', 'e')
print(T1+numbersL) → ERROR
```

6)
```
my_tuples = [(1,2), (5,7,8), (-1,)]
my_tuples[2]= 2
Print(my_tuples) → [(1, 2), (5, 7, 8), 2]
```

7)
```
T1= ('a','d','b','g','e')
print(T1.sort()) → ERROR
print(T1.reverse()) → ERROR
```

8)
```
numbers= [1,4,2,-7,0,6]
rev= numbers.reverse()
print(id(rev) == id(numbers)) → False
```

Test your knowledge (2) - Question

Write the function `methods(L1, L2, n)` that takes as input two lists `L1, L2,` and an integer, `n`.

- The function returns a tuple `T` with the following contents.
- `T` includes all the elements of `L2` and `L1`, concatenated (L2 first).
- The element at position `n` in T must be removed and replaced by the the number 0.
- If `n` is out of the range for `T`, the element in the middle of the tuple must be removed. If the length of the tuple is an even number, then the last number of the first half must be removed. For instance, is the tuple is [1, 2, 3, 4], 2 must be removed, while if the tuple is [1, 2, 3, 4, 5], 3 must be removed.
- The resulting tuple must be returned **sorted in descending order**.
- The function also prints out the length of `T` and the number of times the number `n` appears in the returned list.

Test your knowledge (2) – PsuedoCode / Logic Flow

- **Input:** list **L1,** list **L2,** integer **n**
- **Output:**
  - Return tuple (T) sorted-descending
  - Print T length
  - Print number of times (n) appears in (T)
- **Logic Flow (How to process this input to get the output)**
  - Combine all elements of L1 and L2 (CombinedList)
  - If (n) out of range:
    - If Length of CombinedList is even
      - Remove last number of the first half
    - If Length of CombinedList is odd
      - Remove middle element
  - Else:
    - Element at (n)=0
  - SortedL= Sort Combined-descending
  - Convert SortedL to a tuple (T)
  - Print Length (T)
  - Print number of time n appears in (T)
  - Return T

Solution code

```
def methods(L1, L2, n):

    LCombined= L1+L2 #concatenate the two lists

    lengthL= len(LCombined)  # get length of the combined list

    if(n >= lengthL): # if n is out of range

        if(lengthL%2 !=0): # if the length of combined list is odd
            middle= int(lengthL/2) #round down to get the middle index
            LCombined.pop(middle) # remove the element at middle

        else: # if length of combined list is even
            lastIndxInFirstHalf= int(lengthL/2)-1 # get index of the last element in first half of the
            LCombined.pop(lastIndxInFirstHalf) # remove the elemtent

    else: # if n is within range
        LCombined[n]= 0

    sortedLCombined= sorted(LCombined, reverse=True) # sort the combined List descending order

    T= tuple(sortedLCombined) # convert the sorted list to a tuple

    print('Length of T is: ', len(T))
    print('Number of Times', n, 'Appears in T is: ', T.count(n))
    return T
```

Solution code

```
def methods(L1, L2, n):

    LCombined= L1+L2 #concatenate the two lists

    lengthL= len(LCombined)  # get length of the combined list

    if(n >= lengthL): # if n is out of range

        if(lengthL%2 !=0): # if the length of combined list is odd
            middle= int(lengthL/2) #round down to get the middle index
            LCombined.pop(middle) # remove the element at middle

        else: # if length of combined list is even
            lastIndxInFirstHalf= int(lengthL/2)-1 # get index of the last element in first half of the list
            LCombined.pop(lastIndxInFirstHalf) # remove the element

    else: # if n is within range
        LCombined[n]= 0

    sortedLCombined= sorted(LCombined, reverse=True) # sort the combined List descending order

    T= tuple(sortedLCombined) # convert the sorted list to a tuple

    print('Length of T is: ', len(T))
    print('Number of Times', n, 'Appears in T is: ', T.count(n))
    return T
```