15-110 Principles of Computing – F21

Lecture 13:
Lists 3

Teacher:
Gianni A. Di Caro

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 [])

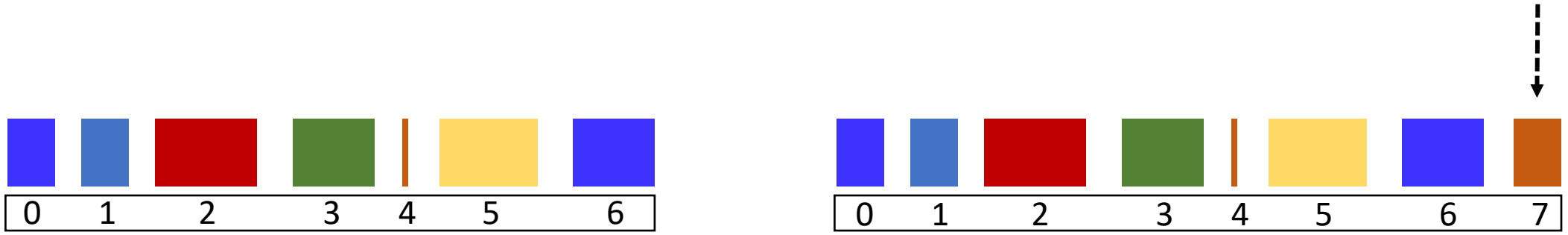
```
primes = [2, 3, 5, 7, 11, 13]
primes = primes + [17]
```

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

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

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

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

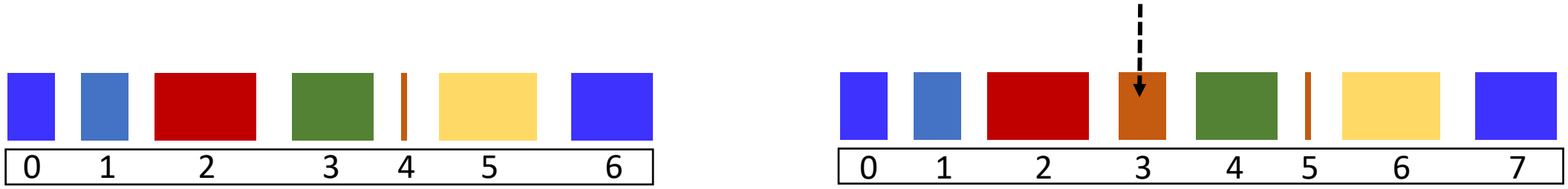



```
primes = [2, 3, 5, 7, 11, 13, 17]
primes.append(19)  →  same list, extended to the end by adding one int literal of value 23

                      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]
```

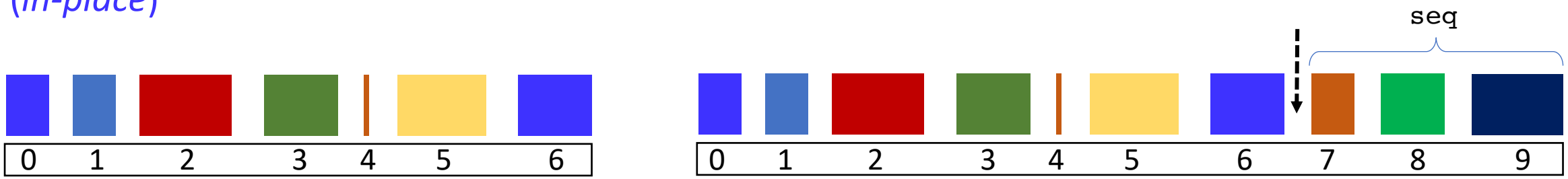
`primes.insert(3,19)` → same list with new item: [2, 3, 5, 19, 7, 11, 13, 17]

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

`primes.insert(0,19)` → same list, with new item, all positions shifted: [19, 2, 3, 5, 19, 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*)



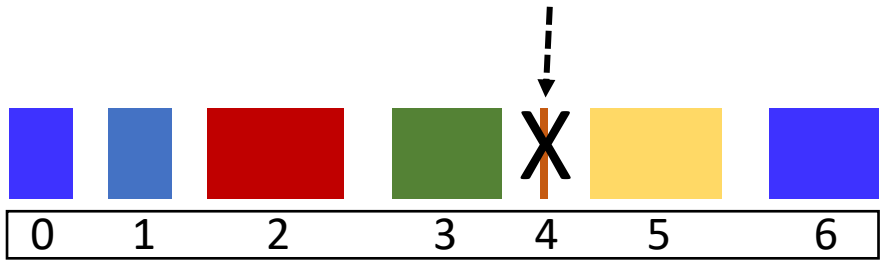

```
primes = [2, 3, 5, 7, 11, 13, 17]
other_primes = (19, 23, 29)
```

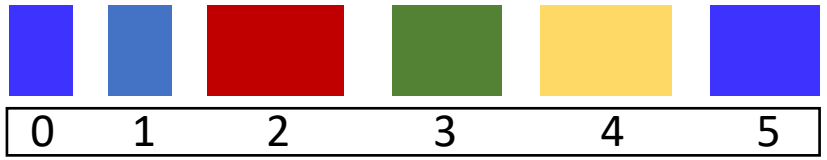
`primes.extend(other_primes)` → same list, extended at the end by adding `other_primes`

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

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**





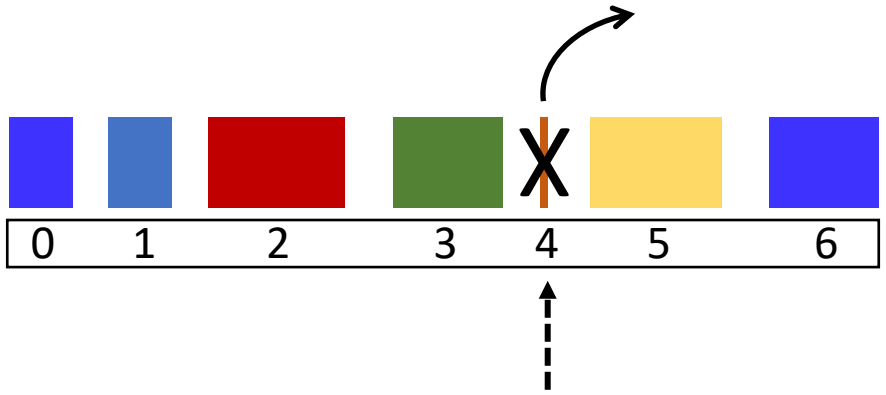

```
numbers = [1, 3, 5, 4, 5, 5, 17]
```

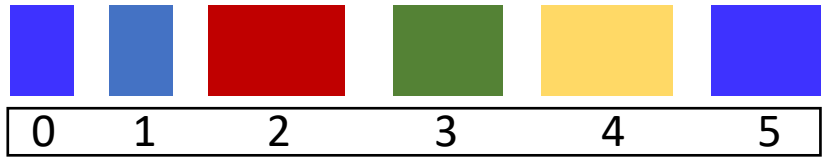
`numbers.remove(5)` → same list, with the first element of value 5 being removed: [1,3,4,5,5,17]

`numbers.remove(15)` → error! an item with value 5 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]

`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()`

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

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

```
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`

`l = (True, False, True).count(True)` → `l` is an integer of value 2 (two occurrences of `True`)

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`

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

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

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

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. …

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)

```
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

```
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?)

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

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

```
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

```
logical = [True, False, True]
n = sum(logical)
```
→ `n` is an integer of value 2

Get a reversed list: [] and slicing (cloning)

- `L.reverse()` changes (in-place) the list `L`

➢ `[::-1]` : Other way to 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 returned / created
  - In-place vs. cloning operations

Reverse the list in-place: `.reverse()` 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*:

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

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

- 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

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)]`

Test your knowledge

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.