



15-110 PRINCIPLES OF COMPUTING – F21

LECTURE 11: LISTS 1

TEACHER:
GIANNI A. DI CARO

Terminology: Scalar vs. Non-Scalar objects

An object type can be:

- **Composite** → made of multiple components,
- Or be **indivisible**

In python this difference is expressed in terms of:

- **Scalar type** (e.g., 1, 3.5, False)
- **Non-scalar (~vector) type** (e.g., 'Toyota', [3, 5, 7])



Vs.



- ✓ **Scalar:** the object is indivisible
- ✓ **Non-scalar:** the object is composed by multiple parts that can be individually manipulated (accessed, modified, removed, added)

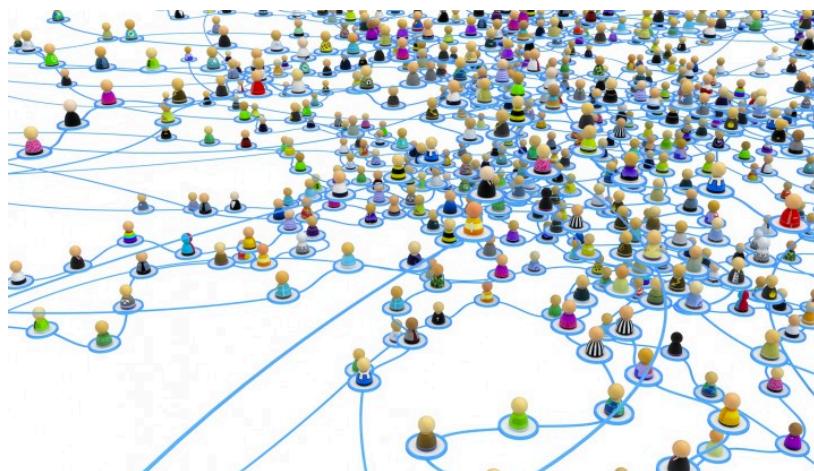
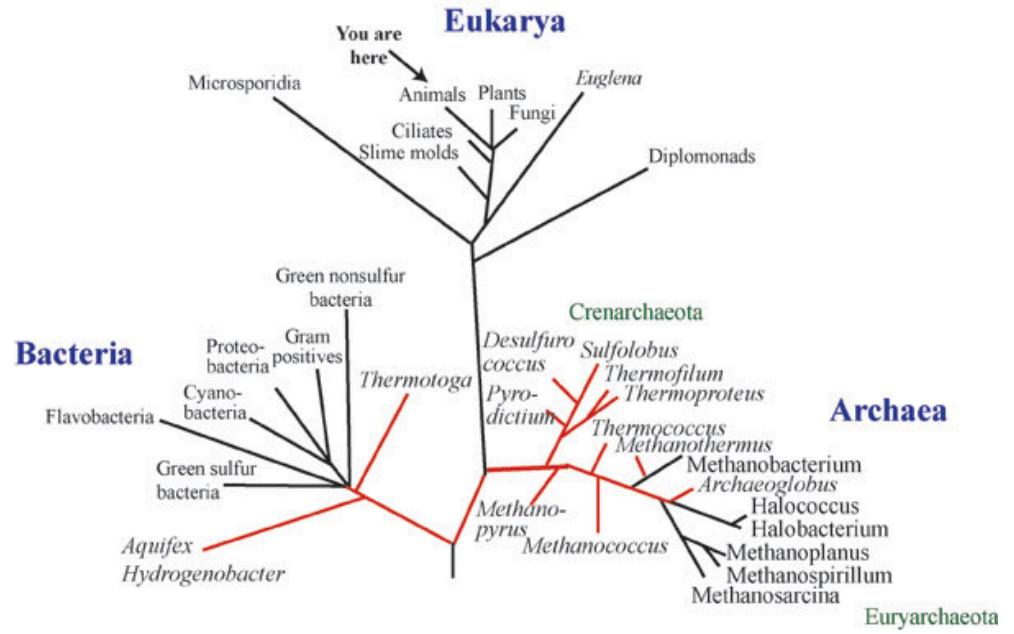
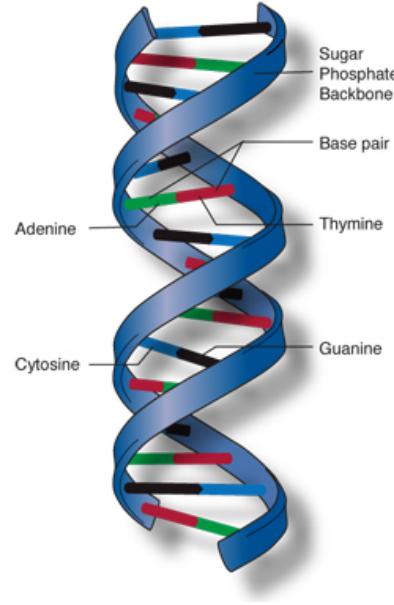
E.g., in the string 'Toyota' I can access and modify the first and last letter to get 'Coyote'

Data structures

- **Data structure**: Container used to store **data** (information) in a specific organized form, a *layout*
- A given *layout* determines a data structure to be efficient in some operations and/or effective in the representation of parts of information, and maybe inefficient / ineffective in others
- → The layout affects how we *access / find / change / add / remove / sort / organize* data



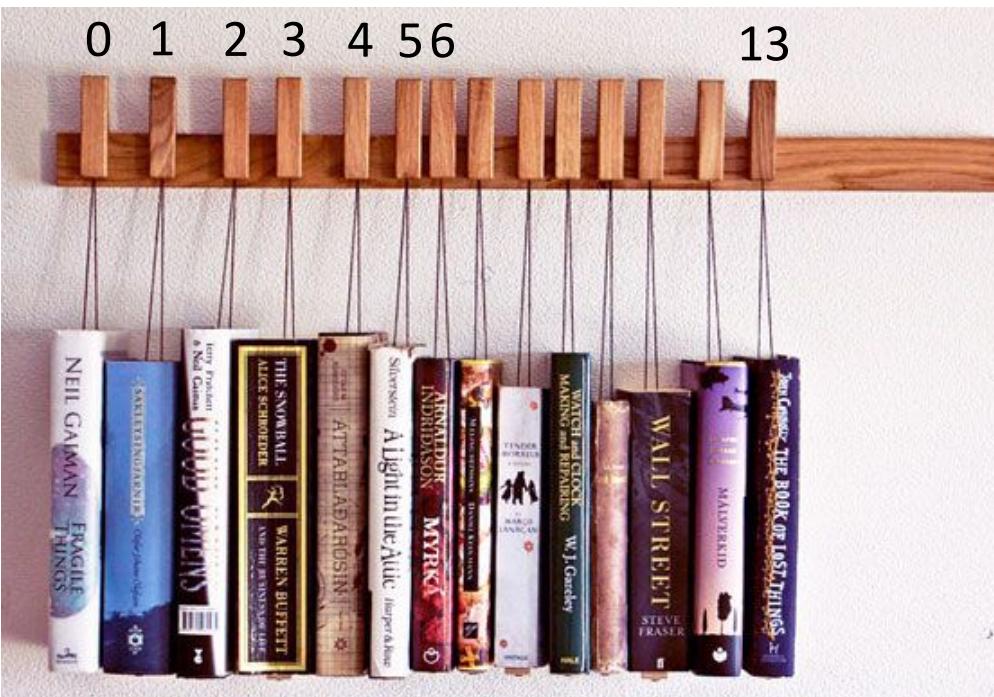
Data structures



List data structure: basic operations

- ❖ List data structure: a container for a **list of objects**,
 - ✓ Each object is placed at a defined **position** in the list
→ List elements are **ordered** by their position

Each book is at a unique position in the list



I can access my list of books in various ways:

- Retrieve the book at *position 11*
- Retrieve the book *Wall Street*
- Which book is at position 5?
- Is *Harry Potter* in the list?
- How many books are in the list
- Is there any empty space in the list rack?

List data type in python

- **List:** (non-scalar type) ordered sequence of objects (any type, not need same type)

- Purpose: *group* items together in the same object
- **Syntax:** [a, b, c, ...,]

- Examples of list variables assigned with list literals

```
prime_numbers = [1, 3, 5, 7, 11]
```

```
irrational_numbers = [2.71828, 3.14159, 1.41421]
```

```
fruits = ['apple', 'pear', 'banana', 'orange']
```

```
person_info = ['Donald', 'Trump', 14, 'June', 1946, 'President']
```

```
fruit_info = ['melon', 3.6, 'yellow', 12.5]
```

```
logical_values = [True, False, True, 255]
```

List data type in python

- Examples of list variables assigned with list literals

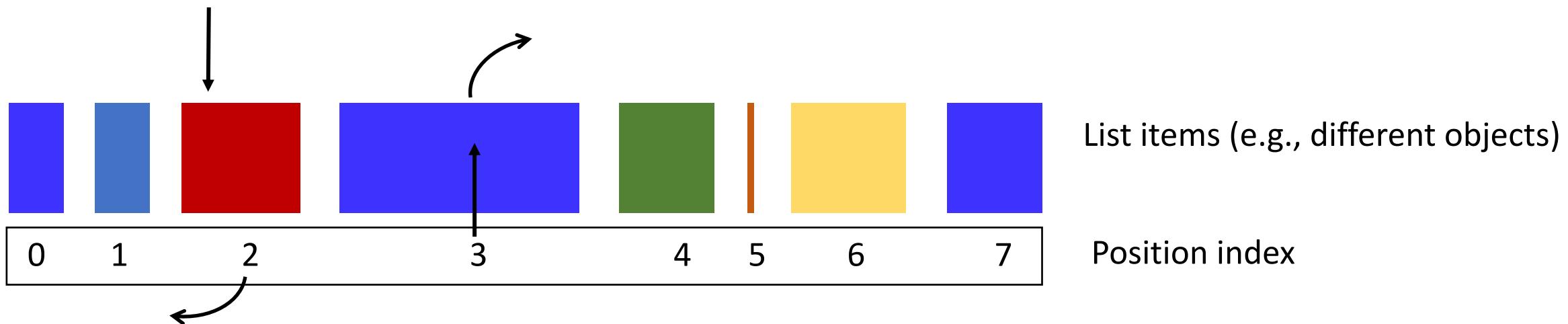
```
empty_list = [ ]
```

```
one_element_list = [ 5 ]
```

```
one_element_list = [ 5, ]
```

```
list_of_lists = [ 1, 'sedan', [ 'Toyota', 'Corolla', 1.8, 2012 ], 52000 ]
```

Basic operations in a list: access / read data



- **Access (read) operations**, at any position:
 - **get data by index** (in 2 steps time)
 - **find data / index by content** (linear number steps for scanning the list)

Basic operations in a list: access / read data with `[]`, `[:]`, `[:]`

- **Syntax**, for a list variable `L`

- *Single list element:* `L[index]` Like `range()`!
- *Subsequence of list elements:* `L[from : to]`
- *Subsequence with a stride:* `L[from : to : step]`

Basic operations in a list: access single elements with []

- **Syntax**, for a list variable L

- Single list element: L[index]
- Index can be positive or negative, or 0

```
prime_numbers = [3, 5, 7, 11, 13]
```

```
even_numbers = [2, 4, 6, 8, 10]
```

```
mix = [-1, 'Hello!', True, 10, 9, 'another string', False]
```

x = prime_numbers[0] → x is of type int and has value 3

x = even_numbers[2] → x is of type int and has value 6

x = prime_numbers[-3] → x is of type int and has value 7

x = mix[1] → x is of type str and has value 'Hello!'

x = mix[2] → x is of type bool and has value True

Basic operations in a list: access subsequences (*slicing*) [:]

- **Syntax**, for a list variable L

- Subsequence of list elements: L[from : to]

```
prime_numbers = [3, 5, 7, 11, 13]
```

```
even_numbers = [2, 4, 6, 8, 10]
```

```
mix = [-1, 'Hello!', True, 10, 9, 'another string', False]
```

x = prime_numbers[0:3] → x is of type list and has value [3, 5, 7]

x = even_numbers[2:4] → x is of type list and has value [6, 8]

x = prime_numbers[-3:-4] → x is of type list and has value []

x = mix[1:4] → x is of type list and has value ['Hello!', True, 10]

x = mix[2:10] → x is of type list and has value [True, 10, 9, 'another string', False]

Basic operations in a list: slicing with a stride [: :]

- **Syntax**, for a list variable L

- Subsequence with a stride: L[from : to : step]

```
prime_numbers = [3, 5, 7, 11, 13]
```

```
even_numbers = [2, 4, 6, 8, 10]
```

```
mix = [-1, 'Hello!', True, 10, 9, 'another string', False]
```

```
x = prime_numbers[0:3:2]           → x is of type list and has value [3, 7]
```

```
x = even_numbers[2:4:5]           → x is of type list and has value [2]
```

```
x = prime_numbers[-1:-3:-1]       → x is of type list and has value [13, 11]
```

```
x = mix[1:4:1]                   → x is of type list and has value ['Hello!', True, 10]
```

```
x = mix[2:10:3]                  → x is of type list and has value [True, 'another string',]
```

Length of a list

- `len(L)` function, that returns an integer value equal to the number of elements in the list

```
x = len([3, 5, 7, 11, 13])
```

```
even_numbers = [2, 4, 6, 8, 10, 12, 14]
```

```
x = len(even_numbers)
```

```
mix = [-1, 'Hello!', True, 10, 9, 'another string', False]
```

```
x = len(mix)
```

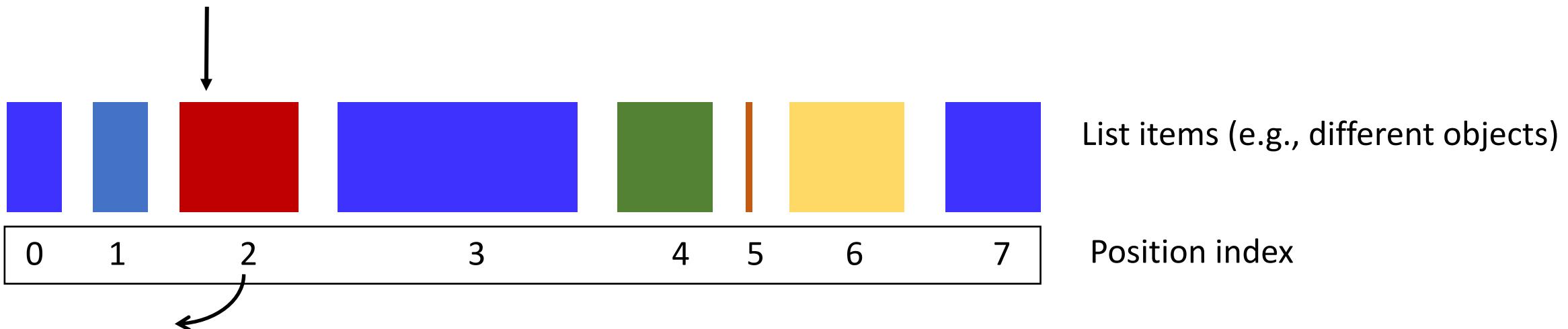
```
x = len([])
```

```
x = len([mix[0]])
```

```
mix = [2, 3, [5, 7, 9], 11 ]
```

```
x = len(mix)
```

Finding data by content



- **Access (read) operations**, at any position:
 - get data by index (in 2 steps time)
 - **find data / index by content** (linear number steps for scanning the list)

Checking presence with membership operators: `in`, `not in`

- Operator `in`: **Membership**, returns `True` if item belongs to the list, `False` otherwise

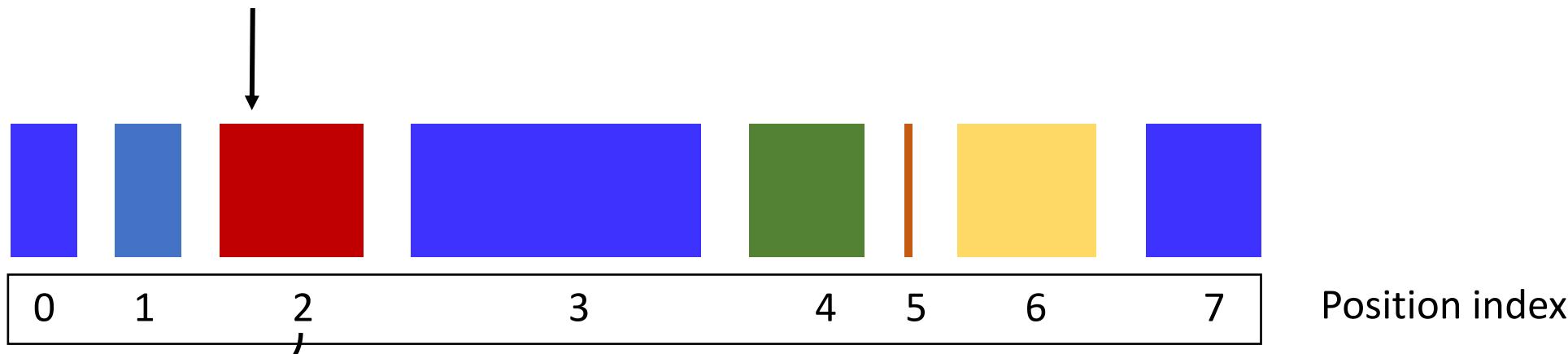
```
prime_numbers = [1, 3, 5, 7, 11]
is_prime = 5 in prime_numbers           → new bool variable with value True
```

```
n = 15
if (n in prime_numbers):
    print(n, 'is a prime number!')
else:
    print(n, 'is not a prime number')
```

- Operator `not in`: **Membership**, returns `False` if item belongs to the list, `True` otherwise

```
prime_numbers = [1, 3, 5, 7, 11]
is_prime = 5 not in prime_numbers        → new bool variable with value False
```

Finding where an element is: `.index()`



- **Access (read) operations**, at any position:
 - get data by index (in 2 steps time)
 - **find** data / index by content
- ❖ **.index(item)** method: return the position index of item

```
L = [1, 4, 7, 0, 9]  
x = L.index(7)
```

→ x has value 2

```
L = [1, 4, 7, 0, 9]  
x = L.index(10)
```

→ Value Error!

Finding where an element is: .index() + in

```
L = [1, 4, 7, 0, 9]  
x = L.index(10)
```

To avoid to crash the program with an error check first with `in`:

```
n = 15  
if (n in L):  
    pos = L.index(n)  
    print(n, 'is at position:', pos)  
else:  
    print(n, 'is not a value in the list')
```

Modify an existing list

- Address the elements to modify using the `[] operator` and change / reassign their value

```
L = [1, 4, 7, 0, 9]           → L is [1, 4, -1, 0, 9]  
L[2] = -1
```

```
L = [1, 4, 7, 0, 9]           → L is [1, 'Doha', -1, 0, 9]  
L[1] = 'Doha'
```

```
L = [1, 4, 7, 0, 9]           → Index Error!  
L[10] = 'Doha'
```

Looping over a list: directly looping over the elements of the list

- ✓ A list is a ... [Sequence!](#)

```
L = [2, 3, 6, -1, 6, 8]
pos_index = 0
for v in L:
    print('List element at position', pos_index,
          'has value', v)
    pos_index += 1
```

- List element at position 0 has value 2
List element at position 1 has value 3
List element at position 2 has value 6
List element at position 3 has value -1
List element at position 4 has value 6
List element at position 5 has value 8

```
L = [2, 3, 6, -1, 6, 8]
for i in range(len(L)):
    print(L[i])
```

Looping over a list: looping over the indices the list

- ✓ A list is a ... **Sequence!**

`range(len(L))` generates the list with all the indices for the elements of L

```
L = [2, 3, 6, -1, 6, 8]
for i in range( len(L) ):
    print('List element at position', i,
          'has value', L[i])
```

`len(L)` is 6

`range(6)` is [0,1,2,3,4,5]

List element at position 0 has value 2
→ List element at position 1 has value 3
List element at position 2 has value 6
List element at position 3 has value -1
List element at position 4 has value 6
List element at position 5 has value 8

Extending a list by adding multiple list elements: +

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

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

```
primes = primes + [19, 23, 29]      → primes it's a new list
```

```
prime_numbers = [1, 3, 5, 7, 11]
```

```
other_primes = [13, 17, 19]
```

```
new_primes = prime_numbers + other_primes → new list with [1,3,5,7,11,13,17,19]
```

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

```
primes = primes + 19                  → Error!
```

tuples

- Lists: [] $L = [3, 5, 7, 11]$
- Tuples: () $L = (3, 5, 7, 11)$

➤ Lists are **mutable** objects: can be changed!

~ similar but

➤ Tuples are **immutable** objects: cannot be changed!

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

$L = (3, 5, 7, 11)$
 $L[2] = -1$ Error!