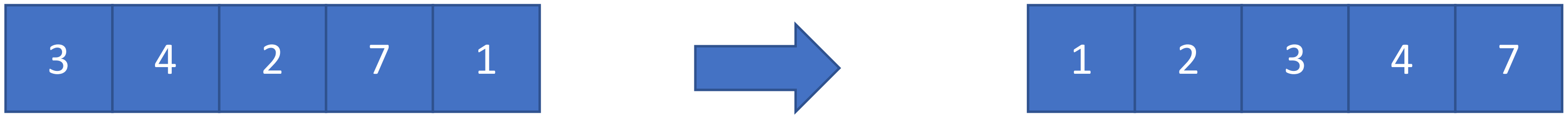
On Sort

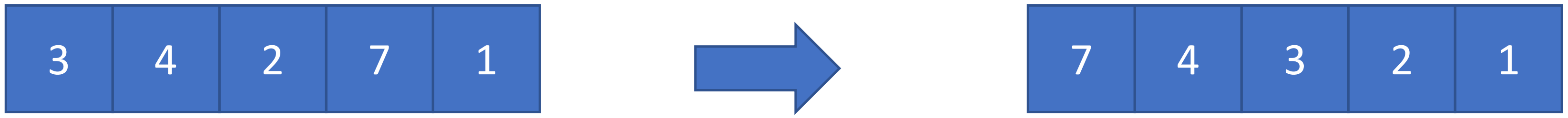
support

Sort

```
L= [3,4,2,7,1]
```



Criteria 1



Criteria 2

Sort…

- Needs
  - Something to order
  - Criteria

- Criteria
  - Default for basic data
    - Numbers, strings, chars, tuples,…
  - Reverse
- Indirect – using criteria
  - Just want a different way to sort
  - must transform into something I know how to sort

Sorted: usual and reverse

```
>>> names = ['Harry', 'Suzy', 'Al', 'Mark']
>>> sorted(names)
['Al', 'Harry', 'Mark', 'Suzy']
>>> sorted(names, reverse=True)
['Suzy', 'Mark', 'Harry', 'Al’]

>>> similar_values = [False, 1, 'A' == 'B', 1 <= 0]
>>> sorted(similar_values, reverse=True)
[1, False, False, False]

>>> numbers = [6, 9, 3, 1]
>>> sorted(numbers, reverse=False)
[1, 3, 6, 9]
```

Sorted: key

There are two main limitations when you're using functions with the key argument.

- the number of required arguments in the function passed to key must be one.
- the function used with key must be able to handle all the values in the iterable.

```
>>> word = 'paper'
>>> len(word)
5
>>> words = ['banana', 'pie', 'Washington', 'book']
>>> sorted(words, key=len)
['pie', 'book', 'banana', 'Washington']

>>> names_with_case = ['harry', 'Suzy', 'al', 'Mark']
>>> sorted(names_with_case)
['Mark', 'Suzy', 'al', 'harry']
>>> sorted(names_with_case, key=str.lower)
['al', 'harry', 'Mark', 'Suzy']
```

Sorted: key

- Can use lambda or function useful to “access” data in tupple, lists, …

```
>>> student_tuples = [
...     ('john', 'A', 15),
...     ('jane', 'B', 12),
...     ('dave', 'B', 10),
... ]
>>> sorted(student_tuples, key=lambda student: student[2])
# sort by age
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
>>> sorted(student_tuples, key=itemgetter(2), reverse=True)
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
```

https://docs.python.org/3/howto/sorting.html

Sorted: Can use lambdas

```
>>> def reverse_word(word):
...     return word[::-1]
...
>>> words = ['banana', 'pie', 'Washington', 'book']
>>> sorted(words, key=reverse_word)
['banana', 'pie', 'book', 'Washington’]

>>> words = ['banana', 'pie', 'Washington', 'book']
>>> sorted(words, key=lambda x: x[::-1])
['banana', 'pie', 'book', 'Washington’]

>>> words = ['banana', 'pie', 'Washington', 'book']
>>> sorted(words, key=lambda x: x[::-1], reverse=True)
['Washington', 'book', 'pie', 'banana']
```

Imagine …

```
L = ["Mario", "Carla", "anabela", "Maria", "nuno"]
```

Mario	Carla	anabela	Maria	nuno

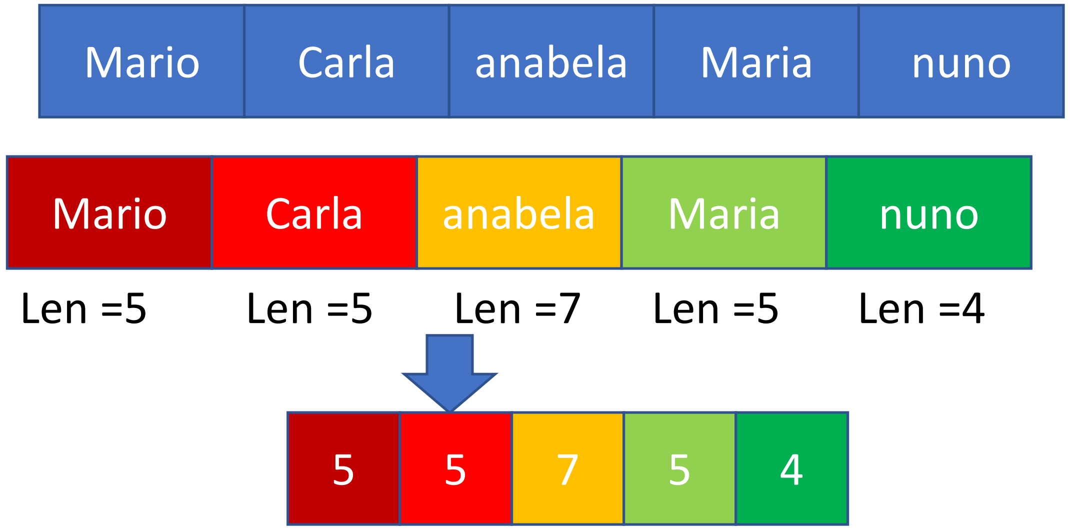
Sorted(L)

Carla	Maria	Mario	anabela	nuno

Imagine …

```
L = ["Mario", "Carla", "anabela", "Maria", "nuno"]
```

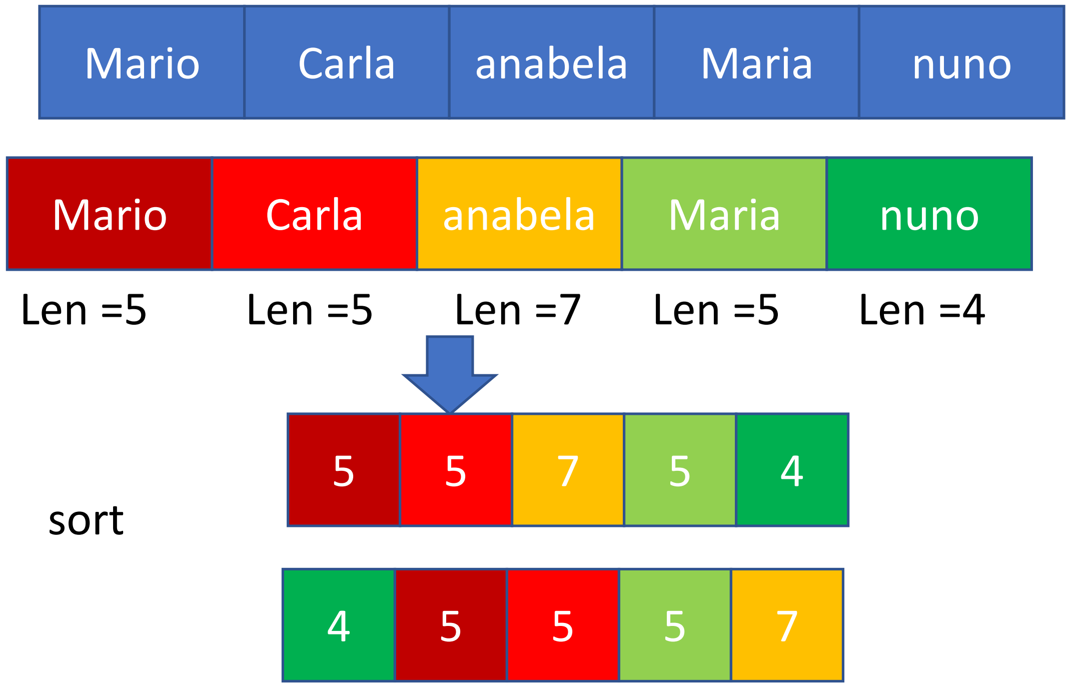
sorted(L, **key**=len)



Imagine …

```
L = ["Mario", "Carla", "anabela", "Maria", "nuno"]
```

sorted(L, **key**=len)

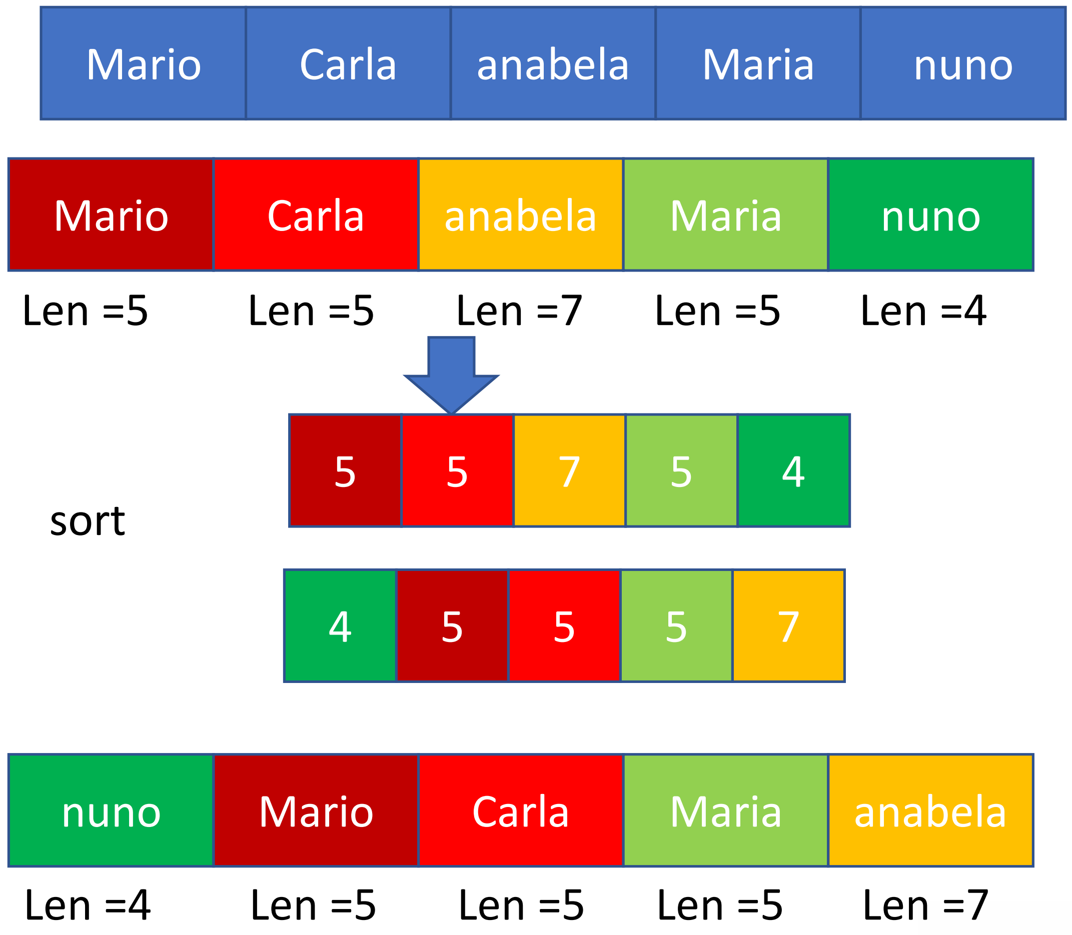


```
['anabela', 'Carla', 'Maria',
'Mario', 'nuno']
```

Imagine …

```
L = ["Mario", "Carla", "anabela", "Maria", "nuno"]
```

sorted(L, **key**=len)

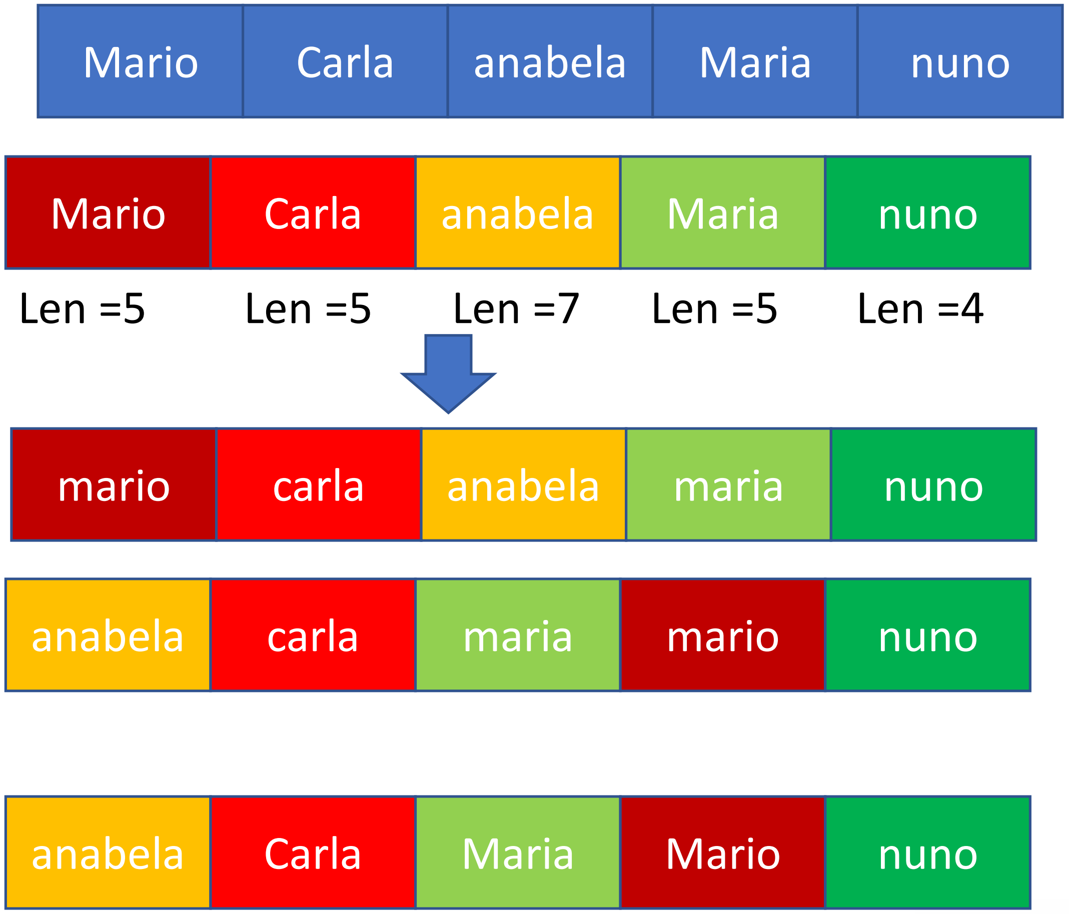


```
['anabela', 'Carla', 'Maria',
'Mario', 'nuno']
```

Imagine …

```
L = ["Mario", "Carla", "anabela", "Maria", "nuno"]
```

sorted(L, **key**=str.lower)

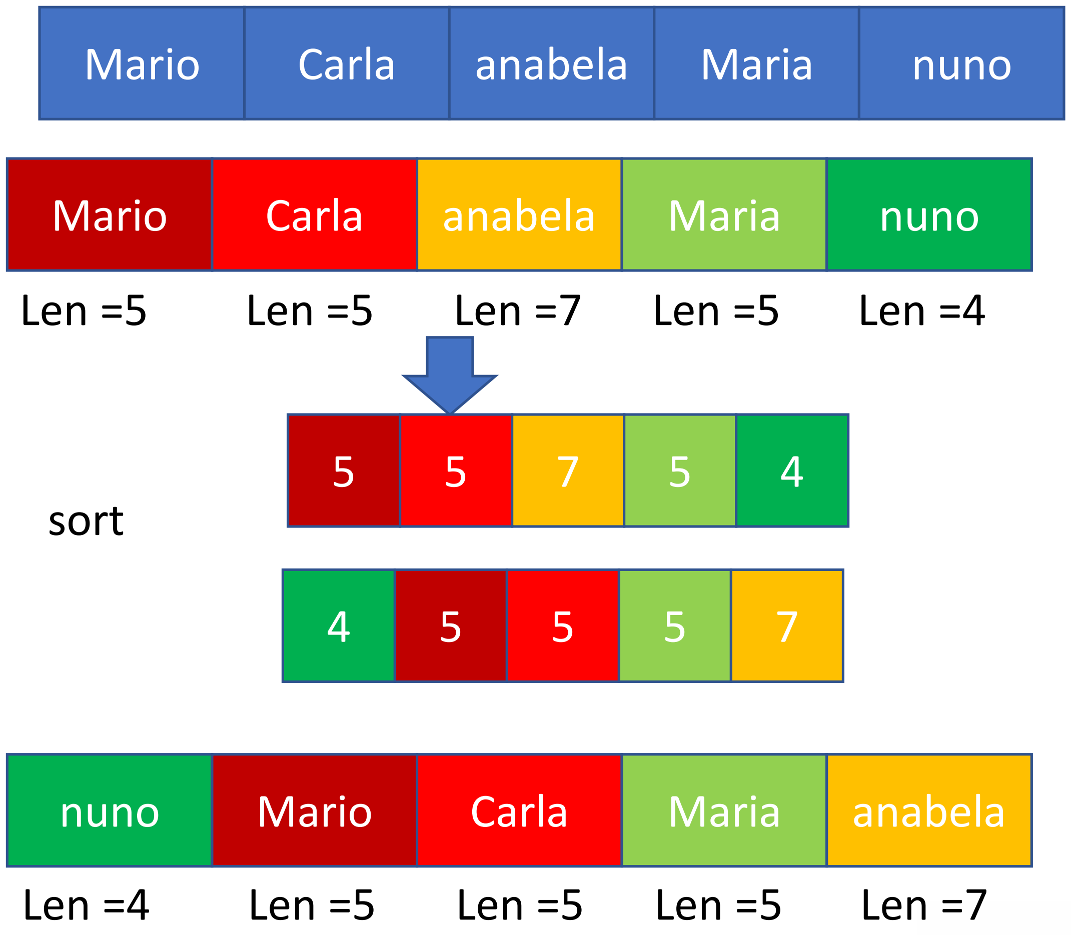


universidade de aveiro

Imagine …

```
L = ["Mario", "Carla", "anabela", "Maria", "nuno"]
```

sorted(L, **key**=len)



And more complex lists???

- The idea is to transform the elements and get something I can sort
- how? Defining functions
- l = [[2, 3], [6, 7], [3, 34], [24, 64], [1, 43]]

```
>>> sorted( l, key=getSecond )
[[2, 3], [6, 7], [3, 34], [1, 43], [24, 64]]
>>> sorted( l, key=getFirst )
[[1, 43], [2, 3], [3, 34], [6, 7], [24, 64]]
```

And more complex lists???

- The idea is to transform the elements and get something I can sort
- how? Defining functions
- l = [[2, 3], [6, 7], [3, 34], [24, 64], [1, 43]]

```
>>> sorted( l, key=getSecond )
[[2, 3], [6, 7], [3, 34], [1, 43], [24, 64]]
>>> sorted( l, key=getFirst )
[[1, 43], [2, 3], [3, 34], [6, 7], [24, 64]]
```

And more complex lists???

```
l = [[2, 3], [6, 7], [3, 34],
[24, 64], [1, 43]]


def getFirst(item):
    return item[0]
sorted( l, key= getFirst)
```

```
#order  [2,6,3,24,1]
[[1, 43], [2, 3], [3, 34],
[6, 7], [24, 64]]
```

```
def getSecond(item):
    return item[1]
sorted( l, key= getSecond)
```

```
# order  [3,7,34,64,43]
[[2, 3], [6, 7], [3, 34], [1,
43], [24, 64]]
```

```
def getStrange(item):
    return item[0]+item[1]
sorted( l, key= getStrange)
```

```
#order [5, 13, 37, 88, 44]
[[2, 3], [6, 7], [3, 34], [1,
43], [24, 64]]
```

Lambda: “easier” functions

```
l = [[2, 3], [6, 7], [3, 34],
[24, 64], [1, 43]]
```

```
~~def getFirst(item):~~
~~    return item[0]~~
sorted( l, key=lambda i:i[0])
```

```
#order  [2,6,3,24,1]
[[1, 43], [2, 3], [3, 34],
[6, 7], [24, 64]]
```

```
~~def getSecond(item):~~
~~    return item[1]~~
sorted( l, key=lambda i:i[1])
```

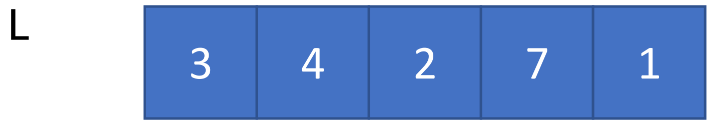
```
# order  [3,7,34,64,43]
[[2, 3], [6, 7], [3, 34], [1,
43], [24, 64]]
```

```
~~def getStrange(item):~~
~~    return item[0]+item[1]~~
sorted( l, key=lambda
i:i[0]+i[1] )
```

```
#order [5, 13, 37, 88, 44]
[[2, 3], [6, 7], [3, 34], [1,
43], [24, 64]]
```

Sort vs sorted

```
L= [3,4,2,7,1]
```

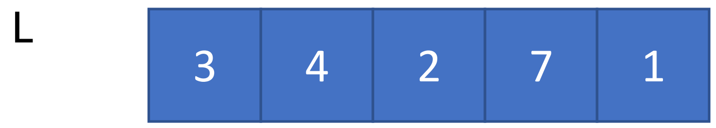


```
# Creates L2. L is not modified!
L2 = sorted(L)
```

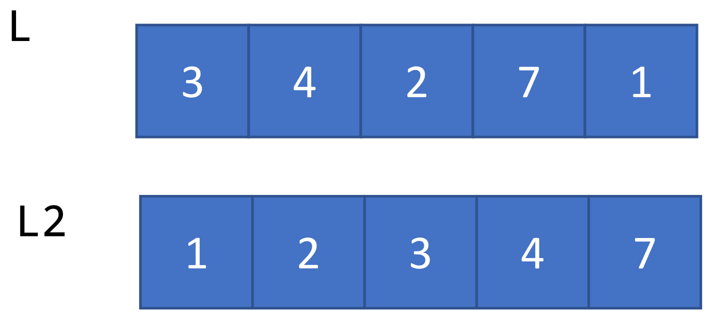
```
# Modifies list L in-place
L.sort()
```

Sort vs sorted

```
L= [3,4,2,7,1]
```



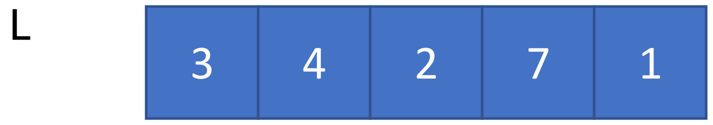
```
# Creates L2. L is not modified!
L2 = sorted(L)
```



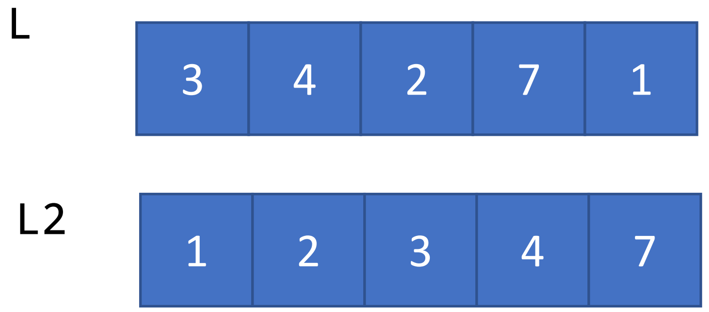
```
# Modifies list L in-place
L.sort()
```

Sort vs sorted

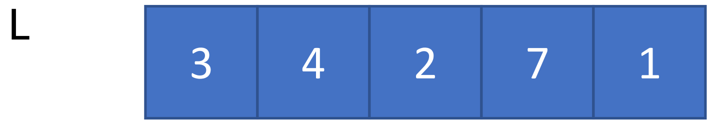
```
L= [3,4,2,7,1]
```



```
# Creates L2. L is not modified!
L2 = sorted(L)
```



```
# Modifies list L in-place
L.sort()
```



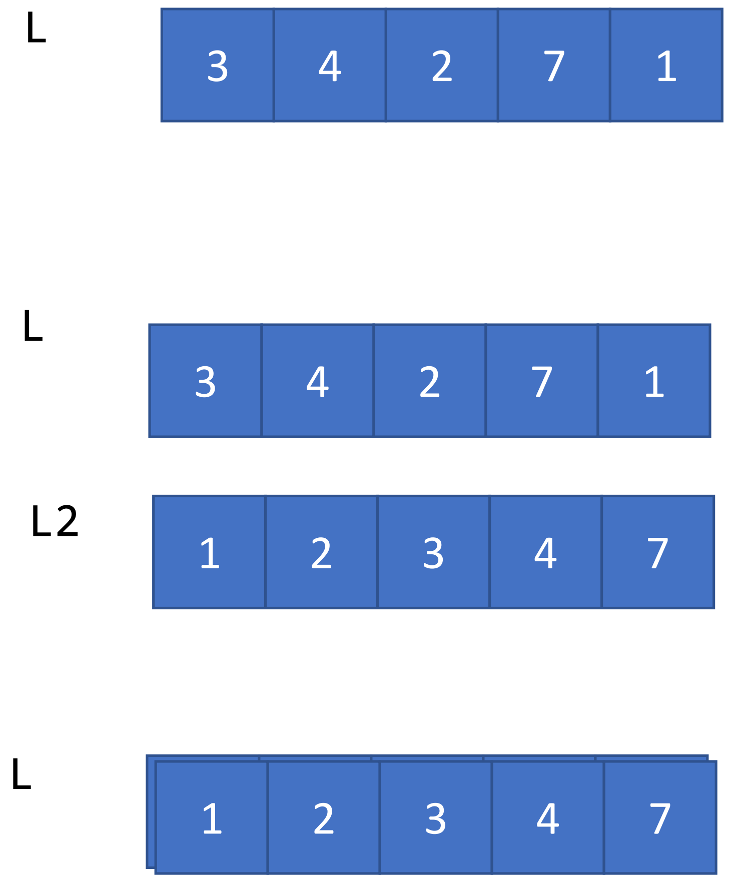
universidade de aveiro

Sort vs sorted

```
L= [3,4,2,7,1]
```

```
# Creates L2. L is not modified!
L2 = sorted(L)
```

```
# Modifies list L in-place
L.sort()
```



Sort vs sorted

```
>>> numbers = [6, 9, 3, 1]
>>> sorted(numbers)
[1, 3, 6, 9]
>>> numbers
[6, 9, 3, 1]
>>> numbers.sort()
>>> numbers
[1, 3, 6, 9]
>>>
```

- With sort there is no way to recover the original list i.e. The initial order

https://realpython.com/python-sort/

Sort vs sorted

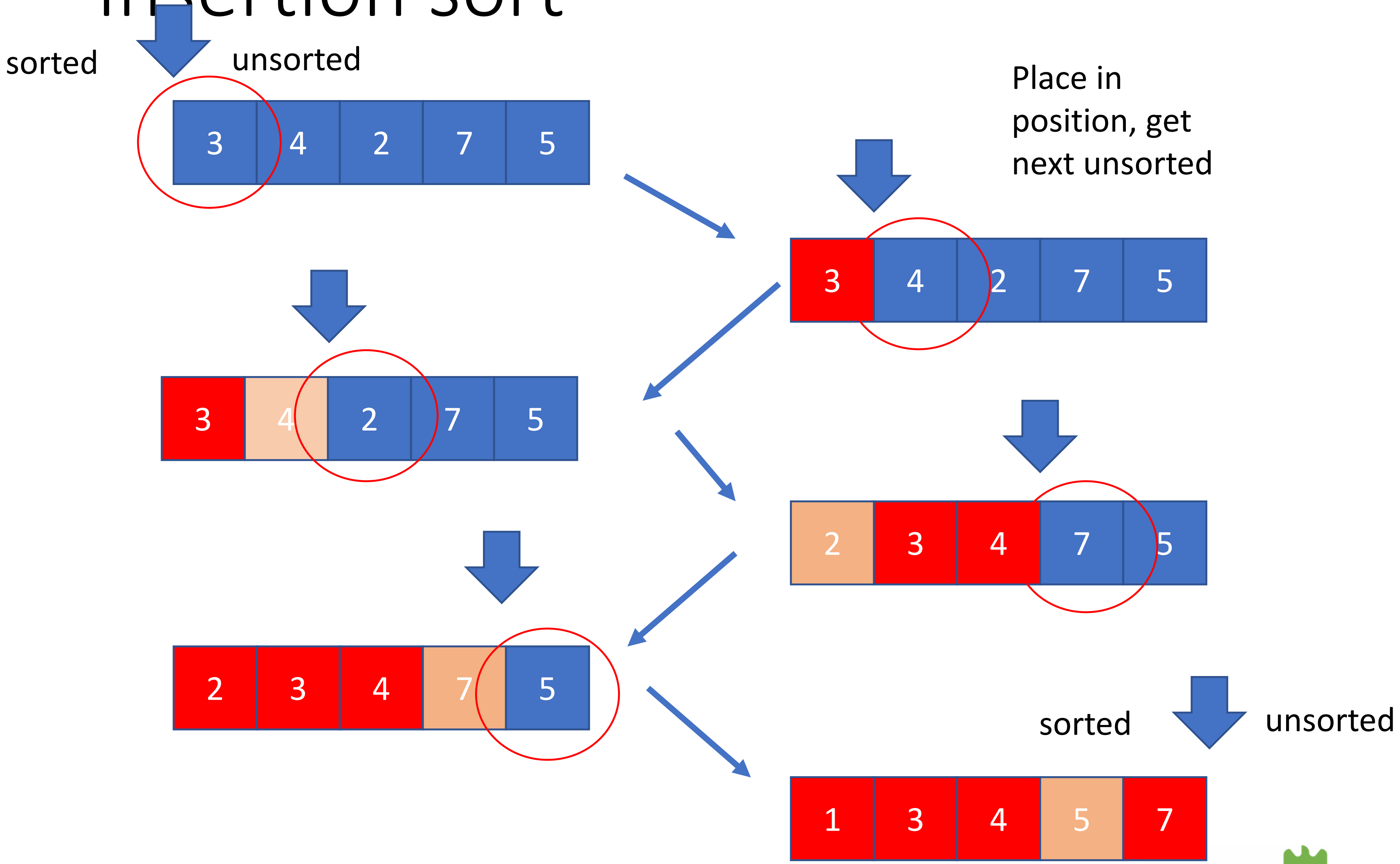
```
>>> phrases = ['when in 
rome', 
...     'what goes around 
comes around', 
...     'all is fair in 
love and war' 
...     ]
>>> phrases.sort(key=lambda 
x: x.split()[2][1], 
reverse=True)
>>> phrases
['what goes around comes 
around', 'when in rome', 
'all is fair in love and 
war']
```

- With sort there is no way to recover the original list i.e. The initial order
- Sort can also have key and reverse parameter

https://realpython.com/python-sort/

insertion

Insertion sort



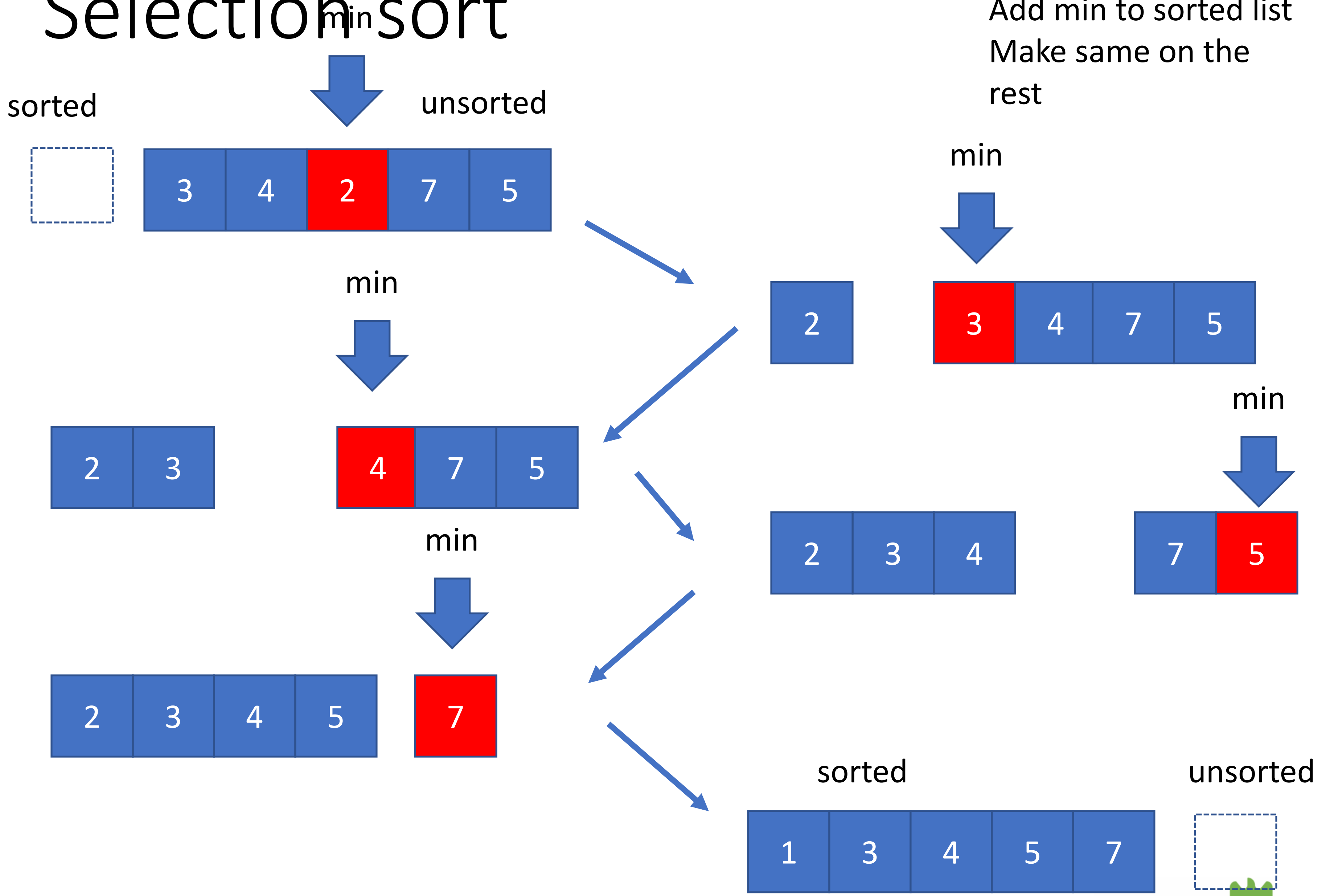
universidade de aveiro

Other option

- Look for maximum / minimum
- Place in result
- Do it for the rest of the list

Selection sort

Find min and rest,
Add min to sorted list
Make same on the rest

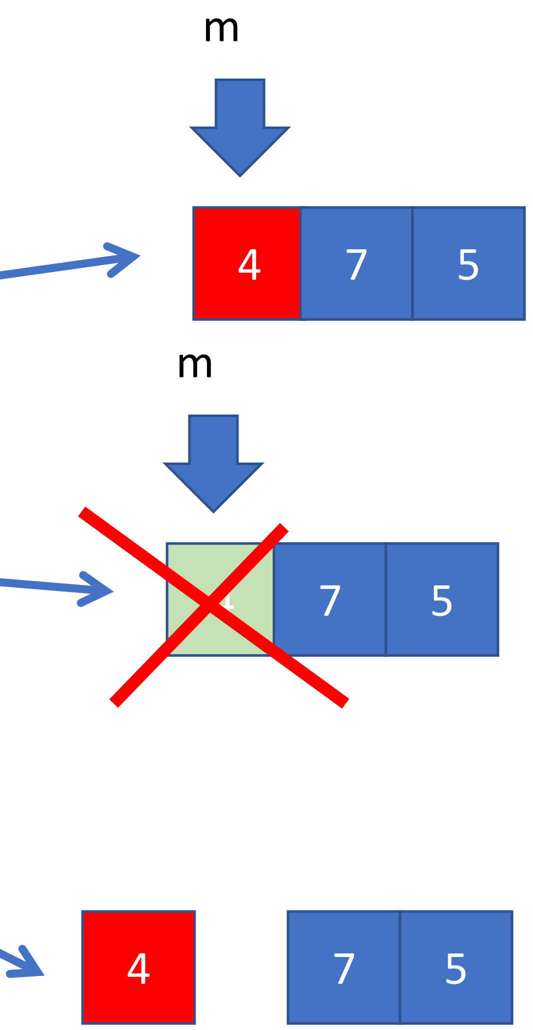


Another option?

```
def mymin( l ):
  m = 0
  for i,y in list(enumerate(l)) :
    if l[m] > y :
       m = i
  min= l[m]
  del l[m]
  return min, l

def mysort_mi( l ):
  l1= l.copy()
  res =[]
  while len(l1)>0 :
    mn, l1= mymin( l1 )
    res.append( mn )
  return res
```

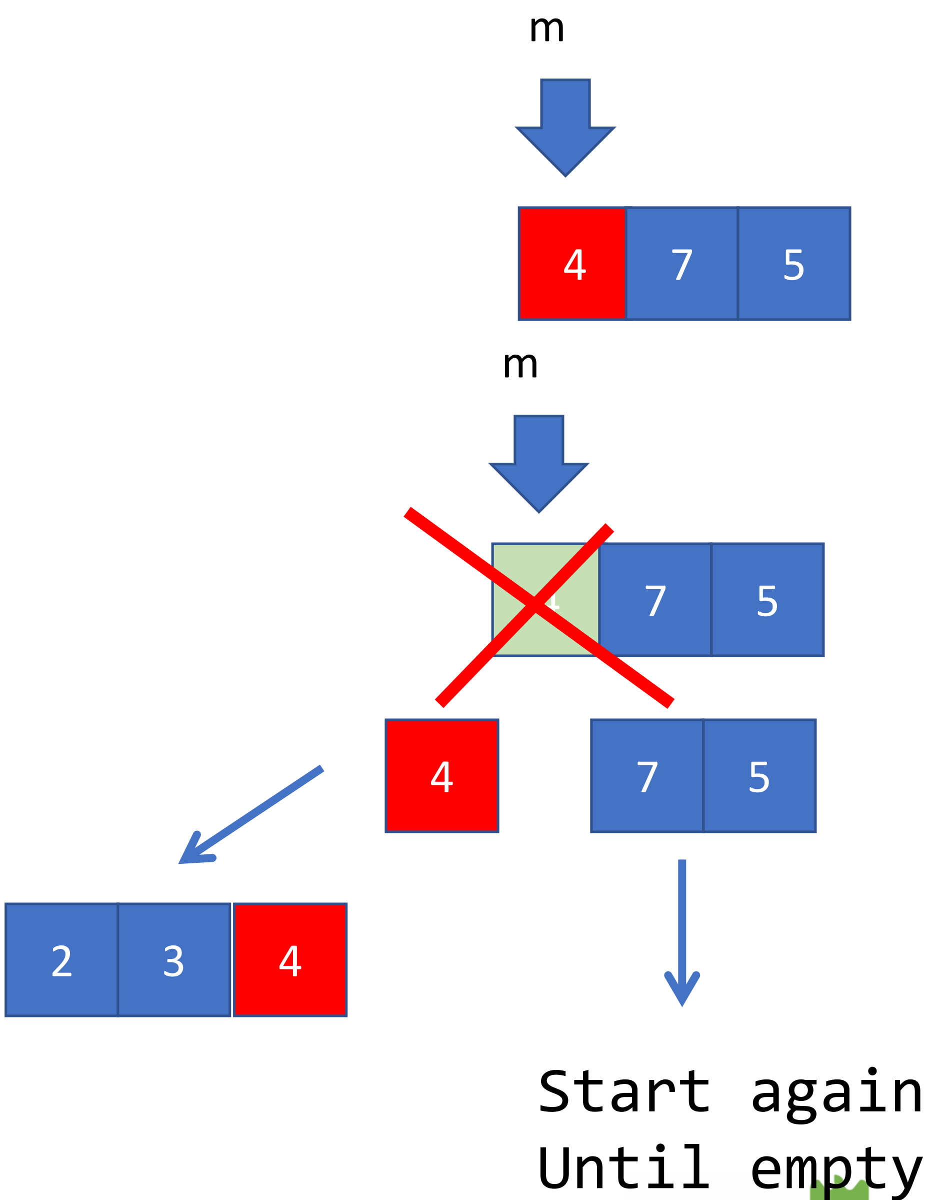
Another option?

```
def mymin( l ):
  m = 0
  for i,y in list(enumerate(l)) :
    if l[m] > y :
      m = i
  min= l[m]
  del l[m]
  return min, l

def mysort_mi( l ):
  l1= l.copy()
  res =[]
  while len(l1)>0 :
    mn, l1= mymin( l1 )
    res.append( mn )
  return res
```



Another option?

```
def mymin( l ):
  m = 0
  for i,y in list(enumerate(l)) :
    if l[m] > y :
       m = i
  min= l[m]
  del l[m]
  return min, l

def mysort_mi( l ):
  l1= l.copy()
  res =[]
  while len(l1)>0 :
    mn, l1= mymin( l1 )
    res.append( mn )
  return res
```



The END