Informator
A SOFRAND COMPANY

SORTING

PYTHON

PYTHON

BASIC SORTING

Lists have their own sort method.

```
a = [1,2,4,6,8]
a.sort()
```

Iterables in general could be sorted using the sorted() function

```
a = [1,2,4,6,8]
sorted(a)
```

list.sort() does an *IN PLACE* sorting

sorted(a) returns a new list that is sorted

PYTHON

KEY FUNCTION

list.sort() & sorted() both got the ”key=function” argument.

”function” is a function that is applied before the comparison is done.

```
list1.sort(key=int)

a = sorted(list1,key=int)

a = sorted(”This is a nice example”.split(),key=str.lower)
```

PYTHON

KEY FUNCTION

Lambda functions are handy.

```
car_tuples = [
        ('Volvo', 'V70', 180),
        ('Tesla', ’Model 3', 138),
        ('Audi', 'A4', 207),
]

# Sort cars by model name
>>> sorted(car_tuples, key=lambda car: car[1])
```

PYTHON

KEY FUNCTION

It also works with instances and attributes

```
class Car(object):
    def __init__(self, name, model , hp):
        self.name = name
        self.model = model
        self.hp = hp

car_objs = [
        Car('Volvo', 'V70', 180),
        Car('Tesla', 'Model 3', 138),
        Car('Audi', 'A4', 207),
]

# Sort cars by model name
>>> sorted(car_objs, key=lambda car: car.hp)
```

PYTHON

EXERCISE

Create a method that returns a sorted iterable with all the nodeConfig-instances in your nodeList.

Sort on the ”numberOfTestsPerformed” attribute.

PYTHON

REVERSE FUNCTION

list.sort() and sorted() has a ”reverse” function so you could chose if you want ascending or descending sorting.

```
# Sort cars by model name
print sorted(car_tuples, key=lambda car: car[1])

# Sort cars by model name Descending
print sorted(car_tuples, key=lambda car: car[1], reverse=True)
```

PYTHON

EXERCISE

Add descending sorting to your previous exercise

PYTHON

SORTING DICTS

```
for key, value in sorted(myDict.items(), key=lambda x: x[0],
reverse=False):
    print key, value, "\n"
```

MORE SORTING

There are more techniques of sorting data structures in general that we haven't mention in this chapter.

Some people loves using decorators. (often called: Decorate-Sort-Undecorate)

Some use the old cmp method.

We also have som modules available for use.

PYTHON

IMPORT OPERATOR

```
from operator import itemgetter, attrgetter


sorted(car_tuple, key=itemgetter(2))

sorted(car_objs, key=attrgetter('hp'))
```

PYTHON

MULTIPLE LEVELS

```
from operator import itemgetter, attrgetter


sorted(car_tuple, key=itemgetter(1,2))

sorted(car_objs, key=attrgetter('hp', 'model'))
```

PYTHON

METHOD CALLER

We could do sorting on the returned value from a specific method.

```
from operator import itemgetter, attrgetter, methodcaller

sorted(car_objs, key=methodcaller(’getEstimatedLifetime'))
```

REVERSING

PYTHON

PYTHON

REVERSING

list.reverse() vs reversed()

PYTHON

REVERSING STRINGS

How do we reverse a string?

list.reverse() method

```
a = list(”Hello World”)
a.reverse()
print a
```

Using str.join() to merge a list to a string

```
a = ””.join(reversed(”Helloooo”))
```

REVERSING STRINGS

Using generator expressions

```
s = ”my string”
a = ''.join((s[i] for i in xrange(len(s)-1, -1, -1)))
```

Using a function with reduce()

```
def reversed_string(s):
    return reduce(lambda a,b : b+a, s)
```

Using slices

```
print ”Hello World”[::-1]
```

PYTHON

WHAT TO CHOSE?

You'll probably find 10 more ways of reversing strings if you start looking.

Functionality vs Readability

```
def reversed_string(s):
    return s[::-1]
```

PYTHON

REVERSED( )

We often use the __reversed__ magic method and the iteration protocol.

__reversed__ should return an iterator that goes backwards.

Use built-in functions as long as possible.

```
def __reversed__(self):
    return reversed(self.text)
```

If __reversed__ is missing it falls back to the ”sequence protocol” using __len__ and __getitem__.

PYTHON

EXERCISE

Add a __reversed__() magic method on your nodeConfigList-object.

Use the reversed() function and for-loop to make sure it's possible to grab all the nodeConfig-instances in reverse order.