Introduction

list comprehensions are a way to define lists. Pros of using list comprehensions include:

1. does not pollute the current scope wwith unwanted variables
2. takes only one line of code

Regarding 1, consider the following example.

In [1]:
```
var = []
for k in range(5):
    var.append(k**2)

print(k)
```

```
4
```

Variable `k` whose only purpose is to build the list still exists after the loop. This is dangerous in case the name `k` is used elsewhere with no initialization.

List comprehensions are made of:

- one or several `for` loops
- a value to fill the container with, possibly dependant from the indexes of the `for` loops
- (optional) a condition `if` / `then` / `else`

Simple examples

Even integers

In [2]:
```
var = [2*k for k in range(10)]
var
```

Out[2]:
```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Even integers that are multiple of 4.

In [3]:
```
var = [2*k for k in range(10) if not 2*k%4]
var
```

Out[3]:
```
[0, 4, 8, 12, 16]
```

Note that with a `else`, location of `if` is also modified.

In [4]:
```
var = [2*k if not 2*k%4 else 0 for k in range(10) ]
var
```

Out[4]:
```
[0, 0, 4, 0, 8, 0, 12, 0, 16, 0]
```

One can use existing other variables:

In [5]:
```
reference = {0: "val_2", 1: "val_1", 2: "val_1", 3: "val_2", 4: "val_1"}
var = [reference[k] for k in range(5)]
var
```

Out[5]:
```
['val_2', 'val_1', 'val_1', 'val_2', 'val_1']
```

With other data containers

list comprehensions can be used with other data containers:

- `generateur`
- `set`
- `dict`
- etc...

Generators

A generator is a data container whose content is not stored into memory until it has to be retrieved. Retrieval is done either using a classical `for`, or the `next` method.

In [6]:
```
var = (letter.upper() for letter in "test" if letter != "e")
var
```

Out[6]:
```
<generator object <genexpr> at 0x79ab557cd970>
```

In [7]:
```
print(next(var))
print(next(var))
print(next(var))
```

```
T
S
T
```

Sets

sets cannot contain duplicate values:

In [8]:
```
var = {k%5 for k in range(100)}
var
```

Out[8]:
```
{0, 1, 2, 3, 4}
```

Dictionaries

In [9]:
```
var = {k: 2*k+1 for k in range(5)}
var
```

Out[9]:
```
{0: 1, 1: 3, 2: 5, 3: 7, 4: 9}
```