

Comprehension

Formula

```
l = []  
for {var} in {collection of items}:  
    l.append({expression})
```

Equivalent list comprehension

```
l = [{expression} for {var} in {collection of items}]
```

Example

```
l = []  
for x in range(10):  
    l.append(x*x)
```

Equivalent list comprehension

```
l = [ x*x for x in range(10) ]
```

Conditional comprehension

Formula

```
l = []  
for {var} in {collection of items}:  
    if {condition}:  
        l.append({expression})
```

Equivalent list comprehension

```
l = [{expression} for {var} in {collection of items} if {condition}]
```

Example

```
l = []  
for x in [5,6,7,8,9,10,11]:  
    if x%2 == 0:  
        l.append(x*x)
```

Equivalent list comprehension

```
l = [ x*x for x in [5,6,7,8,9,10,11] if x%2 == 0 ]
```

List comprehension

```
l = [ x*x for x in range(10) if x%2 == 0 ]  
l = list( x*x for x in range(10) if x%2 == 0 )
```

Dictionary comprehension

```
d = { x:x*x for x in range(10) if x%2 == 0 }  
d = dict( { x:x*x for x in range(10) if x%2 == 0} )
```

The lecture video was missing two braces. You need this for the second method.



Set comprehension

```
s = { x*x for x in range(10) if x%2 == 0 }  
s = set( x*x for x in range(10) if x%2 == 0 )
```

Tuple comprehension (Tuple comprehension requires "tuple()")

```
t = tuple( x*x for x in range(10) if x%2 == 0 )
```


How to study Python, knowing that everything in python is an object?

Case study: Lists

```
m = [1, 2, 3]
or
m = list([1,2,3])
```

Class object: **list**

Attributes:

Data attributes :
(all hidden)

Methods:

(visible or hidden)

`__setitem__()`

`__getitem__()`

`append()`

`pop()`

⋮
⋮
⋮

Instance object of **list** class: **m**

Unique non-method
attributes: (all hidden)

Link to data
(also hidden)

Common **list** attributes:

linked to all the
class object
attributes including
Methods
(hidden or visible)

We can't directly touch it!!!
Data is protected!!!
We only have methods!!!

`[1, 2, 3]`

Programming is all about

- **Data**
- **Data manipulation**

In python, making data is easy. Manipulating the data is **NOT** easy. We must use class methods that are specifically designed to manipulate the data.

```
m.__setitem__(idx, value)
m.__getitem__(idx)
m.append(value)
m.pop(idx)
m.insert(idx, value)
```

```
list.__setitem__(m, idx, value)
list.__getitem__(m, idx)
list.append(m, value)
list.pop(m, idx)
list.insert(m, idx, value)
```

Instance object of the **list** class: **m**

Unique non-method
attributes: (all hidden)

Link to data
(also hidden)

Common **list** attributes:

linked to all the
class object
attributes including
Methods
(hidden or visible)

[1, 2, 3]

There are some syntactic sugars for built-in data types (or classes) in Python.

`int, float, complex, str`

`list, tuple, dict, set`

Operators: `[], +, -, *, /, %, **, in, not, is, and, or, >, >=, <, <=, ==, !=, etc...`

Not all operators are available for all types. It depends on whether it makes sense and if it is convenient for that data type.

Comprehension: `list, dictionary, set, tuple`

Note: Syntactic sugars are as important as the key syntaxes. And, often, it is more than sugar. It is not critical to *understand* the language. But, without them, Python won't be as powerful. Ex: a new walrus operator, `:=`. If you're curious, see <https://realpython.com/lessons/assignment-expressions/>

How should we study each class (or, object/data type)?

1. Check out how to *initialize* the object. There may be multiple ways.
2. Read examples to understand *methods* to manipulate the data
3. Check if there are any syntactic Kool-Aid, such as useful *operators*.

Ex: **Dictionary**

1. Initialization of object:

```
d = { 'k':34, 'm':'abc', 'NYC':'cold' } # there are more ways.
```

2. Methods for data manipulation: (getting a list of methods: `dir(d)`)

```
d.update( { 'SB' : '69F' } )  
m_val = d.pop( 'm' )  
keys = d.keys()           # etc...
```

3. Syntactic sugar:

```
NYC_temp = d[ 'NYC' ]      # same as d.__getitem__('NYC')  
new_d = { x*x for x in range(10) } # comprehension
```


How about non-native data types?

(Python allows programmers to use operators to behave whatever way they want them to behave. We will see how operators behave differently in other non-native data types such as **array** and **dataframe**.)

The approach to study 3rd-party classes (or other classes in the standard library) is the same.

Ex: **Seq** class of Biopython

1. Initialization of an object (i.e., initializing the data):

```
seq = Seq('atgcatgc') # There are more ways.
```
2. Methods for data manipulation: This is an immutable object. Thus, we can't manipulate it!! Instead, we get a modified version of the data as a return 'object'. (See **MutableSeq** class) Use `dir(seq)` to get a list of methods.

```
ex_seq = seq.upper()           # A new Seq object is returned.  
mRNA_seq = seq.transcribe()    # Again, a new Seq object  
rev_seq = seq.reverse_complement() # etc...
```
3. Syntactic sugar:

```
sub_seq = seq[2:5]             # Returns a new seq object with partial seq
```

Do we need to memorize all of them?

No. (That's why exams will be fully open book.)

Worth memorizing (to be efficient and effective):

Control flow syntaxes (**if-elif-else**, **for-loop**, **while-loop**)

Syntaxes for **functions** and **classes**

Most methods and operators of **str**, **list**, **tuple**, and **dictionary**

A few selected methods (e.g., **reverse_complement**, **translate**, etc) of **Seq** and **SeqRecord**,

Useful tool functions, such as, **dir()**, **type()**, **help()**, **callable()**, **%time**, **%debug**, etc

(**git pull**, **add**, **commit**, **push**, though not Python.)

