Python: List/Dictionary Comprehension

1405 Instructor: Ruiqing (Sam) Cao

List Comprehension

- An **iterable** is any data object that can be iterated over (i.e., returned one at a time, or looped over with a `for` or `while` loop)

  e.g., `list`, `dict`, `str`, `tuple`, `set`

- List comprehension generates a new list in a concise way by applying an expression to *each element* of an existing *iterable*

- Syntax: `[expression for x in iterable if condition]`

  → The "`if condition`" part is optional

Iterable vs. Non-Iterable

- An *iterable* type is any data object that can be iterated over (i.e., returned one at a time, or looped over with a `for` or `while` loop)

  e.g., `list`, `dict`, `str`, `tuple`, `set`

➢Non-scalars are not necessarily iterable, though a lot of them are

- A *non-iterable* type is the opposite of an iterable type

➢All scalars in Python are technically non-iterable

➢Strings are a bit ambiguous: a string represent an atomic, indivisible value (e.g., in the database context it satisfies 1NF), but a string in Python is technically non-scalar (characters) and iterable as well

Exercises: List Comprehension

1. Create a new list that equals element-wise sum of two lists `li1` and `li2` of the same length (Basic)
2. Create a new list that consists of integer elements from an existing list `li` (Filter)
3. Turn a list `li` of integers into 5-digit zip codes of string type with 0s in the beginning to fill space (hint: use f-string :05) (Function)
4. Create a Boolean mask for whether each element in a list `li` of integers is even (True) or odd (False) (Condition)
5. Flatten a nested list `li`, i.e., combine the elements of the nested list `li` (which are themselves lists) into a large list (Nested)

Exercises: List Comprehension (Basic)

(Basic) Create a new list that equals element-wise sum of two lists `li1` and `li2` of the same length

```
new_li= [li1[j]+li2[j] for j in range(len(li1))]
```

Exercises: List Comprehension (Filter)

(Filter) Create a new list that consists of integer elements from an existing list `li`

```
new_li= [x for x in li if type(x)==int]
```

Exercises: List Comprehension (Function)

(Function) Turn a list `li` of integers into 5-digit zip codes of string type with 0s in the beginning to fill space (hint: use f-string :05)

```
new_li= [f'{x:05}' for x in li]
```

Exercise: List Comprehension (Condition)

(Condition) Create a Boolean mask for whether each element in a list `li` of integers is even (True) or odd (False)

```
new_li= [True if x%2==0 else False for x in li]
```

Exercises: List Comprehension (Nested)

(Nested) Flatten a nested list `li`, i.e., combine the elements of the nested list `li` (which are themselves lists) into a large list

```
new_li= [x for sublist in li for x in sublist]
```

Mapping: Element-Wise Transformation

- Similar to list comprehension, mapping is another common way to apply element-wise transformation to an iterable

- Basic syntax: `map(function,iterable)`

- Example: Turn all elements in a list `li` into strings

```
list(map(str,li))
```

Mapping: Element-Wise Transformation

- Similar to list comprehension, mapping is another common way to apply element-wise transformation to an iterable

- Basic syntax: `map(function,iterable1,iterable2)`

- Example: adding numeric elements of two equal-length lists

```
list(map(lambda a,b: a+b, [1,2,3], [-1,-2,-3]))
```

Exercises: Mapping

(Mapping) Create a Boolean mask for whether each element in a list li is greater than 5

Exercises: Mapping

(Mapping) Create a Boolean mask for whether each element in a list li is greater than 5

```
list(map(lambda x: True if x>5 else False, li))
```

Dictionary Comprehension

- Very similar to list comprehension, generates a new dictionary by applying key and value expressions to *each element* of an existing *iterable*

- Syntax: `{key_expr:value_expr for x in itrb if cond}`

→ The "`if cond`" part is optional

Exercises: Dictionary Comprehension

1. Create a dictionary that maps each element in the list `li` into 1 plus that element (Basic)
2. Create a dictionary that maps only integer item in an existing list `li` into 1 plus that element (Filter)
3. Create a dictionary that maps unique values of the list `li` to their number of occurrences (hint: use `set()`) (Function)
4. Create a dictionary that maps each item in a list `li` to a Boolean mask for whether it is integer (`True`) or not (`False`) (Condition)
5. Create a dictionary using the list of `keys` and the list of `values` corresponding to each key (hint: use `zip()`) (Zipping)

Exercise: Dict Comprehension (Basic)

(Basic) Create a dictionary that maps each element in the list `li` into 1 plus that element

```
new_dict= {x:x+1 for x in li}
```

Exercise: Dict Comprehension (Filter)

(Filter) Create a dictionary that maps only integer item in an existing list `li` into 1 plus that element

```
new_dict= {x:x+1 for x in li if type(x)==int}
```

Exercises: Dict Comprehension (Function)

(Function) Create a dictionary that maps unique values of the list `li` to their number of occurrences (hint: use `set()`)

```
new_dict= {x:li.count(x) for x in set(li)}
```

Exercise: Dict Comprehension (Condition)

(Condition) Create a dictionary that maps each item in a list `li` to a Boolean mask for whether it is integer (`True`) or not (`False`)

```
new_dict= {x:type(x)==int for x in li}
```

Exercise: Dict Comprehension (Zipping)

- (Zipping) Create a dictionary using the list of `keys` and the list of `values` corresponding to each key (hint: use `zip()`)

```
new_dict = dict(zip(keys,values))
```

instead of

```
new_dict = {keys[i]:values[i] for i in range(len(keys))}
```