# Python Tips and Tricks: Takeaways ⤴

## Syntax

### LIST COMPREHENSIONS

- Converting a for loop to a list comprehension:
  - Using a for loop:

    ```python
    letters=['a', 'b', 'c', 'd']

    caps=[]

    for l in oletters:

        caps.append(l.upper())
    ```

  - Using a List comprehension:

    ```python
    caps = [l.upper() for l in letters]
    ```

- Common list comprehension patterns:
  - Transforming a list

    ```python
    ints = [25, 14, 13, 84, 43, 6, 77, 56]

    doubled_ints = [i * 2 for i in ints]
    ```

  - Creating test data

    ```python
    tenths = [0.0 + i/10 for i in range(5)]
    ```

  - Reducing a list

    ```python
    big_ints = [i for i in ints if i>= 50]
    ```

### NESTED LIST COMPREHENSIONS

- Converting a nested for loop to a nested loop comprehension:
  - Using a for loop:

    ```python
    data = [

            [3, 2, 4],

            [2, 8, 6],

            [1, 2, 1],

            ]

    times ten = []

    for row in data:

        new_row = []

        for item in row
    ```

## PRODUCING AN ITERABLE OF INTEGERS

- Generating a sequence of integers:

  - Using a nested list comprehension:

```
range(1,6) # equiv of [1, 2, 3, 4, 5]
```

- Generating a sequence with a 'skip' of

```
2
```

  :

```
range(0,10,2) # equiv of [0, 2, 4, 6, 8]
```

## LOOPING THROUGH MULTIPLE LISTS

- Enumerate over a list to index a second list:

```
list_1 = [1, 24, 9, 18, 12]

list_2 = [1, 8, 3, 9, 4]

for i, item_1 in enumerate(list_1):

    item_2 = list_2[i]

    print(item_1, item_2) # outputs pairs of values from both lists
```

- Iterating over two or more lists at once:

```python
lower = ['a', 'b', 'c']

upper = ['A', 'B', 'C']

z = zip(lower, upper) # equiv of [('a', 'A'), ('b', 'B'), ('c', 'C')]
```

## LAMBDA FUNCTIONS

- Converting a definition to a lambda function:
    - Defining a function:

    ```python
    def double(x):

        return x * 2
    ```

    - Defining a lambda function:

    ```python
    run_function(function=lambda x: x * 2)
    ```

## ANY & ALL

- Return

```python
True
```

if the boolean value of at least one item in iterable is

```python
True

any(iterable)
```

- Return

```python
True
```

if the boolean value of all items in iterable are

```python
True

any(iterable)
```

## DICTIONARY COMPREHENSIONS

- Creating a dictionary comprehension:
    - Using a for loop

    ```python
    l = [

        ["Julie", 27],

        ["Alex", 12],

        ["Bruno", 50],

        ]

    d = {}
    ```

```
for name, score in l:

    d[name] = score

print(d) # {"Julie": 27, "Alex": 12, "Bruno": 15}
```

- Using a dictionary comprehension

```
d = {name:score for name, score in l}

print(d) # {"Julie": 27, "Alex": 12, "Bruno": 15}
```

## Concepts

- A list comprehension provides a concise way of creating lists using a single line of code, where:
  - You start with an iterable object
  - Optionally Transform the items in the iterable object
  - Optionally reduce the items in the iterable object using an if statement
  - Create a new list
- A dictionary comprehension provides a concise way of creating dictionaries using a single line of code, similar to a list comprehension with lists
- The `range()` function returns an iterable containing a sequence of integers
- Lambda functions can be defined in a single line, which lets you define a function at the time you need it
- Any/All:

| Function | Explanation | Equivalent |
|----------|-------------|------------|
| `all([a, b, c])` | Returns True if the boolean value of **every** value is True, otherwise returns False | `bool(a) and bool(b) and bool(c)` |
| `any([a, b, c])` | Returns True if the boolean value of **at least one** value is True, otherwise returns False | `bool(a) or bool(b) or bool(c)` |

# Resources

- [Python Documentation: List Comprehensions](#)
- [Python Documentation: Dictionary Comprehensions](#)
- [Python Documentation: Lambda Functions](#)