

# List and Dictionary Comprehension

Code	Explanation
<code>[output for i in iterable if condition]</code>	<b>List comprehension</b> output – expression for what to put in the list i – takes items from the iterable one by one iterable – a collection of objects condition – a condition used to filter out elements of the resulting list
<code>{key:val for i in iterable if condition}</code>	<b>Dictionary comprehension</b> key – expression for keys to put in the dictionary val – expression for values to put in the dictionary i – takes items from the iterable one by one iterable – a collection of objects condition – a condition used to filter out elements of the resulting dictionary
<code>[out1 if condition else out2 for i in iterable]</code>	<b>List comprehension (with if else)</b> out1 – expression for what to put in the list if condition is true out2 – expression for what to put in the list if condition is false i – takes items from the iterable one by one iterable – a collection of objects condition – a condition used to adjust elements of the resulting list
<code>{key:val1 if condition else val2 for i in iterable}</code>	<b>Dictionary comprehension (with if else)</b> out1 – expression for what value to put in the dict if condition is true out2 – expression for what value to put in the dict if condition is false i – takes items from the iterable one by one iterable – a collection of objects condition – a condition used to adjust elements of the resulting list

# Regular Expression

## re MODULE FUNCTIONS

<code>re.findall(A,B)</code>	Matches all the instances of an expression A in a string B and returns them in a list
<code>re.search(A,B)</code>	Matches the first instance of an expression A in a string B, and returns it as a re match object
<code>re.split(A,B)</code>	Split a string B into a list using a delimiter A
<code>re.sub(A,B,C)</code>	Replace A with B in the string C

## SPECIAL CHARACTERS

<code>^</code>	Matches the expression to its right at the start of a string. It matches every such instance before each <code>\n</code> in the string.
<code>\$</code>	Matches the expression to its left at the end of a string. It matches every such instance before each <code>\n</code> in the string.
<code>.</code>	Matches any character except the line terminators like <code>\n</code>
<code>\</code>	Escapes special characters or denotes character classes
<code>A B</code>	Matches expression <code>A</code> or <code>B</code> . If <code>A</code> is matched first, <code>B</code> is left untried.
<code>+</code>	Greedily matches the expression to its left 1 or more times
<code>*</code>	Greedily matches the expression to its left 0 or more times
<code>?</code>	Greedily matches the expression to its left 0 or 1 times. But if <code>?</code> is added to a quantifier (like <code>+</code> , <code>*</code> , or <code>?</code> itself) it will perform matches in a non-greedy manner.
<code>{m}</code>	Matches the expression to its left m times, and not less.
<code>{m,n}</code>	Matches the expression to its left m to n times, and not less

## CHARACTER CLASSES

<code>\w</code>	Matches alphanumeric characters, which means <code>a-z</code> , <code>A-Z</code> , and <code>0-9</code> . It also matches the underscore <code>_</code>
<code>\d</code>	Matches digits, which means <code>0-9</code>
<code>\D</code>	Matches any non-digits
<code>\s</code>	Matches whitespace characters, which include the <code>\t</code> , <code>\n</code> , <code>\r</code> , and space characters
<code>\S</code>	Matches non-whitespace
<code>\b</code>	Matches the boundary (or empty string) at the start and end of a word, that is between <code>\w</code> and <code>\W</code>
<code>\B</code>	Matches where <code>\b</code> does not, that is, the boundary of <code>\w</code> characters
<code>\A</code>	Matches the expression to its right at the absolute start of the string whether in single or multi-line mode
<code>\Z</code>	Matches the expression to its left at the absolute end of a string whether in single or multi-line mode

## SETS

<code>[]</code>	Contains a set of characters to match
<code>[amk]</code>	Matches either <code>a</code> , <code>m</code> , or <code>k</code> . It does not match <code>amk</code>
<code>[a-z]</code>	Matches any alphabet character from <code>a</code> to <code>z</code>
<code>[a\ -z]</code>	Matches <code>a</code> , <code>-</code> , or <code>z</code> . It matches <code>-</code> because <code>\</code> escapes it.
<code>[a-] or [-a]</code>	Matches <code>a</code> or <code>-</code> , because <code>-</code> is not being used to indicate a series of characters
<code>[a-z0-9]</code>	Matches any alphabet character <code>a-z</code> or numeric character <code>0-9</code>
<code>[ (+* ) ]</code>	Special characters become literal inside a set, so this matches <code>(</code> , <code>+</code> , <code>*</code> , and <code>)</code>
<code>[^ab5]</code>	Adding <code>^</code> excludes any characters in the set. So this matches characters that are not <code>a</code> , <code>b</code> , or <code>5</code> .

## GROUPS

<code>()</code>	Matches the expression inside the parentheses and groups it
<code>(?:A)</code>	Matches the expression represented by A
<code>A(?:B)</code>	Positive lookahead: Matches the expression A only if it is followed by the expression B
<code>A(?:!B)</code>	Negative lookahead: Matches the expression A only if it isNOT followed by the expression B
<code>(?&lt;=B)A</code>	Positive lookbehind: Matches the expression A only if the expression B is right before it (left)
<code>(?&lt;!B)A</code>	Negative lookbehind: Matches the expression A only if the expression B is NOT right before it

# NumPy

## CREATING ARRAYS

<code>np.zeros((3,4))</code>	Create a 3 by 4 array of zeros
<code>np.ones((2,3,4),dtype=np.int)</code>	Create a 2 by 3 by 4 array of ones with int type
<code>np.arange(10,25,5)</code>	Create an array of evenly spaced values between 10 and 25 step by 5
<code>np.linspace(0,2,9)</code>	Create an array of 9 evenly spaced values between 0 and 2 (inclusive)
<code>np.full((2,2),7)</code>	Create a constant 2 by 2 array
<code>np.eye(2)</code>	Create a 2 by 2 identity matrix
<code>np.random.rand((2,2))</code>	Create 2 by 2 array of random values between 0 and 1
<code>np.random.randint(1,10,3)</code>	Create array of length 3, of random integers between 1 and 10 (exclusive)
<code>np.random.randn((2,2))</code>	Create 2 by 2 array of random values drawn from standard normal

## INSPECTING ARRAYS

<code>arr.shape</code>	Check how many elements in each dimension
<code>arr.size</code>	Check total number of elements in array
<code>arr.ndim</code>	Check the number of dimensions
<code>len(arr)</code>	Check the length of the array
<code>arr.dtype</code>	Check the type of each element in the array
<code>arr.astype(int)</code>	Change the type of the elements to type integer

## AGGREGATE FUNCTIONS

<code>arr.sum()</code>	Sum of all the elements in the array
<code>arr.min()</code>	Minimum of all the elements in the array
<code>arr2d.max(axis=1)</code>	Maximum value of each row
<code>arr.argmax()</code>	Get the index of the maximum element in the array
<code>arr.cumsum()</code>	Cumulative sum of all elements in the array
<code>arr.mean()</code>	Mean of all elements in the array
<code>arr.median()</code>	Median of all elements in the array
<code>arr.std()</code>	Standard deviation of all elements in the array
<code>arr.quantile(p)</code>	The pth quantile of all elements in the array

## COPYING ARRAYS

<code>arr.view()</code>	Creates a view of an array (also happens you create a new array with slicing)
<code>arr.copy()</code>	Creates a copy of an array

## SUBSETTING/SLICING/INDEXING

<code>arr[2]</code>	Select the 3rd element (element at index 2)
<code>arr2d[1,2]</code>	Select element in 2nd row and 3rd column (row index 1 and column index 2)
<code>arr[start:stop:step]</code>	Select elements from index <code>start</code> to index <code>stop</code> (exclusive) stepping by <code>step</code>
<code>arr2d[start:stop:step, start:stop:step]</code>	Select row and columns elements from index <code>start</code> to index <code>stop</code> (exclusive) stepping by <code>step</code>
<code>arr[::-1]</code>	Reverse the array
<code>arr[arr&lt;2]</code>	Boolean indexing: select elements that are less than 2

## ARRAY MANIPULATION

<code>arr.ravel()</code>	Flatten the array
<code>arr.reshape(new_ax0,new_ax1)</code>	Reshape but don't change the data
<code>np.sort(arr)</code>	Sort an array
<code>np.append(arr1,arr2)</code>	Append elements of <code>arr2</code> to <code>arr1</code>
<code>np.insert(arr1,1,5)</code>	Insert the element 5 into index one of <code>arr1</code>
<code>np.delete(a,1)</code>	Delete the element at index 1 of <code>arr1</code>
<code>np.vstack((arr1,arr2))</code>	Stack arrays vertically (row-wise)
<code>np.hstack((arr1,arr2))</code>	Stack arrays horizontally (column-wise)

# Pandas

## INSPECTING SERIES/DATAFRAME

<code>df.dtypes</code>	Check data types of each column of dataframe
<code>df.shape</code>	Check number of rows and columns
<code>df.index</code>	Check the row labels of a dataframe (or labels of Series)
<code>df.columns</code>	Check column labels of a dataframe
<code>df.info()</code>	Check info about dataframe (e.g. dtypes and memory etc)

## SELECTING ELEMENTS

<code>df.a</code> or <code>df['a']</code>	Select column labeled 'a'
<code>df.iloc[0,1]</code>	Indexing by position row index 0 and column index 1
<code>df.iloc[start:stop:step,start2:stop2:step2]</code>	Slicing by position (rows before comma, columns after)
<code>df.loc[0,'a']</code>	Indexing by label row label 0 and column label 'a'
<code>df.loc[start:stop:step,start2:stop2:step2]</code>	Slicing by label (rows before comma, columns after)
<code>df[df['a']&gt;12]</code>	Boolean indexing: select rows where elements in column 'a' are greater than 12
<code>df[df['b'].str.contains('pattern')]</code>	Boolean indexing: selecting rows where elements in column 'b' matches "phrase"

## AGGREGATE FUNCTIONS (mostly the same as NumPy arrays)

<code>df.idxmax()</code>	Get the index (label) of the maximum element
<code>df.idxmin()</code>	Get the index (label) of the minimum element