# List and Dictionary Comprehension

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

# Regular Expression

## `re` MODULE FUNCTIONS

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

## SPECIAL CHARACTERS

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

## CHARACTER CLASSES

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

## SETS

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

### GROUPS

| | |
|---|---|
| `()` | Matches the expression inside the parentheses and groups it |
| `(?:A)` | Matches the expression represented by `A` |
| `A(?=B)` | Positive lookahead: Matches the expression `A` only if it is followed by the expression `B` |
| `A(?!B)` | Negative lookahead: Matches the expression `A` only if it isNOT followed by the expression `B` |
| `(?<=B)A` | Positive lookbehind: Matches the expression `A` only if the expression `B` is right before it (left) |
| `(?<!B)A` | Negative lookbehind: Matches the expression `A` only if the expression `B` is NOT right before it |

# NumPy

### CREATING ARRAYS

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

### INSPECTING ARRAYS

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

### AGGREGATE FUNCTIONS

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

### COPYING ARRAYS

| | |
|---|---|
| `arr.view()` | Creates a view of an array (also happens you create a new array with slicing) |
| `arr.copy()` | Creates a copy of an array |

### SUBSETTING/SLICING/INDEXING

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

### ARRAY MANIPULATION

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

# Pandas

### INSPECTING SERIES/DATAFRAME

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

### SELECTING ELEMENTS

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

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

| | |
|---|---|
| `df.idxmax()` | Get the index (label) of the maximum element |
| `df.idxmin()` | Get the index (label) of the minimum element |