

### Comprehensions

List:

```
name of new list = [expression for item in iterable if condition == True]
```

```
squares = [number**2 for number in numbers if x < 5]
```

Generators:

```
use () not []
```

```
print(next(result))
print(next(result))
```

### Dictionaries

```
for x, y in art_galleries.items():
    print(x)
    print(y)
# x with return keys,
# y values
```

### Set Functions to process Iterable Objects

Create sets from a list:

```
cookies_eaten_today = ['chocolate chip', 'peanut butter', ..., 'chocolate chip', 'oatmeal cream', 'chocolate chip']
```

```
types_of_cookies_eaten = set(cookies_eaten_today)
```

Adding elements to a set:

```
.add() adds single elements types_of_cookies_eaten.add('biscotti')
```

### Set Functions to process Iterable Objects (cont)

```
.update() merges in another set or list types_of_cookies_eaten.update(cookies_we_will_eat)
```

Removing:

```
.discard() safely removes an element from the set by value types_of_cookies_eaten.discard('biscotti')
```

Combining Sets:

```
.union() returns a set of all the unique values
```

```
cookies_jason_ate.union(cookies_hugo_ate)
```

```
.intersection() method identifies overlapping data
```

```
cookies_jason_ate.intersection(cookies_hugo_ate)
```

```
.difference() method identifies data present in the set on which the method was used that is not in the arguments (-) cookies_jason_ate.difference(cookies_hugo_ate)
```

### Lambda Functions

Syntax:

```
LambdaFunctionName = arguments : expression
```

```
DefineFunction = lambda (param1, paramn: param1 ** paramn)
```

### Using a Lambda Function inside another Function

```
# a function that always doubles the number you send in
def myfunc(n):
    return lambda a : a * n
mydoubler = myfunc(2)
print(mydoubler(11))
```

### Lambda with Map

```
# Create a list of strings:
spells = ["pr ote go", " acc io", " expecto patron um", " leg ili men s"]
# Use map() to apply a lambda function over spells: shout_spells
shout_spells = map(lambda item: item + '!!!', spells)
# Convert shout_spells to a list: shout_spells_list
shout_spells_list = list(shout_spells)
# Print the result
print(shout_spells_list)
```

### Reduce

```
# Import reduce from functools
from functools import reduce
# Create a list of strings: stark
stark = ['robb', 'sansa', 'arya', 'brandon', 'rickon']
```



By datamansam

[cheatography.com/datamansam/](https://cheatography.com/datamansam/)

Published 3rd September, 2022.

Last updated 3rd September, 2022.

Page 1 of 2.

Sponsored by **CrosswordCheats.com**

Learn to solve cryptic crosswords!

<http://crosswordcheats.com>

### Reduce (cont)

```
> # Use reduce() to apply a lambda function
over stark: result
result = reduce(lambda item1, item2: item1
+ item2, stark)
# Print the result
print(result)
```

### Filter

```
nums = [1, 2, 3, 4, 5, 6, 7, 8,
9, 10]
print( " Ori ginal list of
intege rs: ")
print( nums)
print( " \Results less than 3
when divided by 2 from the said
list:")
LessThan3 = list(f ilt er( -
lambda x: x//2 < 3, nums))
print( Les sThan3)
```

### Iterating through DataFrame Columns

```
# Extract column from DataFrame:
col
col = df[col_name]

# Iterate over each column in
DataFrame
for entry in col:
    action
```

### Iterating through DataFrames

```
# Define count_entries()
def count_ent ries(df,
col_name= 'la ng'):
    """ Return a
dictionary with counts of
occ urre nces as value
for each key."""
    # Initialize an empty
dictio nary: cols_count
```

### Iterating through DataFrames (cont)

```
> cols_count = {}
# Add try block
try :
    # Extract column from DataFrame: col
col = df[col_name]

    # Iterate over each column in
DataFrame
for entry in col:

    # If entry is in cols_count, add 1
if entry in cols_count.keys():
        cols_count[entry] += 1
    # Else add the entry to cols_count,
set the value to 1
else:
        cols_count[entry] = 1

    # Return the cols_count dictionary
return cols_count
# Add except block
except:
    pass
# Call count_entries(): result1
result1 = count_entries(tweets_df, 'lang')
# Print result1
print(result1)
```

### apply, applymap and map

Apply:	Applymap:	Map:
to apply a function along the axis of a dataframe,	element wise operation across one or more rows and columns of a dataframe.	Substitutes the series value from the lookup dictionary, Series or a function

DFs and Series	Only Dataframes	Used only for a Series object
----------------	-----------------	-------------------------------

Applied to both series and elements	Applied to elements individually	Applied to series
-------------------------------------	----------------------------------	-------------------

### Code Eamples of apply, applymap and map

```
df.apply(np.sum, axis=0)
-> col sums
df.app ly( np.sum, axis=1)
-> row sums
df.app lym ap( lambda x: x**2)
-> Every df element squared
s = pd.Ser ies ([' cat', 'dog',
np.nan, 'rabbit'])
s.map( {'cat': 'kitten', 'dog':
'puppy'})
```



By **datamansam**

[cheatography.com/datamansam/](http://cheatography.com/datamansam/)

Published 3rd September, 2022.

Last updated 3rd September, 2022.

Page 2 of 2.

Sponsored by **CrosswordCheats.com**

Learn to solve cryptic crosswords!

<http://crosswordcheats.com>