Cheatography

Data Flow with Python

by datamansam via cheatography.com/139410/cs/32160/

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(r-esult))	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.upd-ate(cookies _ we_will_ eeat)

Removing:

.discard() safely removes an element from the set by value	types _ of _ cookies _ eaten.discar-d('biscotti')

Combining Sets:

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

cookies _jason _ ate.union(cookies _ hugo _ ate)

.intersection() method identies overlapping data

cookies _jason _ ate.intersection(cookies _ hugo _ ate)

.difference() method identies data present in the set on which the method was used that is not in the arguments (-)	cookies _jason _ ate.differen-ce(cookies _ hugo _ ate)

Lambda Functions

Syntax:

LambdaFun-ctionName =	arguments : expression
DefineFun-ction =	lambda (param1, paramn: param1 ** paramn)

Using a Lamda 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( myd oub ler (11))
```

Lambda with Map

```
# Create a list of strings:
spells
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_ spe lls _list
shout_ spe lls _list = list(s -
hou t_s pells)
# Print the result
print( sho ut_ spe lls _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/

Published 3rd September, 2022.
Last updated 3rd September, 2022.

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( " \Resuls 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 rie s(df, col_na me= 'la ng'):
        " " " Return a dictionary with counts of
        occ urr ences as value for each key."""
       # Initialize an empty dictio nary: cols_count
        col s_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_c ount, add 1
                       if entry in cols_c oun t.k eys():
                                                -
col s_c oun t[e ntry] += 1
                       # Else add the entry to cols_c ount, set the value to 1
                        else:
                                                -
col s_c oun t[e ntry] = 1

               # Return the cols_count dictionary
                return cols_count
       # Add except block
        except:
               pass
# Call count_ ent ries():
result1
result1 = count_ ent rie s(t - wee ts_df, 'lang')
# Print result1
print( res ult1)
```

Iterating through DataFrames (cont)

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'})
```