Introduction to Python

Fondren Library
Research Data Services

FLAGS WITH WHILE LOOPS

- Sometimes it makes more sense to create a flag to signal to Python to break the while loop.
- Programs will run while the flag is set to True.

```
prompt = "Input a message. I will repeat it until you type 'quit'"
message = ""


active = True
while active:
    message = input(prompt)

    if message = 'quit':
        active = False

    else:
        print(message)
```

```
prompt = "Input a message. I will
repeat it until you type 'quit'"
message = ""
active = True



while active:
   message = input(prompt)

   if message : 'quit'

   active = False

else:
       print (message)
```

```
user_input = ""
while user_input.lower()
!= "exit":
   user_input =
input("Type 'exit' to
end the loop:  ")
```

a side note on while loops

- while loops make for great techniques when you want to search through data while some condition is true
- let's look at an example

```
number = 1
while number <= 5:
    print(number)
    number += 1
```

```
1
2
3
4
5
```

while loops with a flag

- while loops can react to a flag (like an on-off switch)

```
prompt = "Input a message. I will repeat it until you type 'quit'"
message = ""


active = True
while active:
    message = input(prompt)

    if message = 'quit':
        active = False

    else:
        print(message)
```

aggregating data

- the **groupby** function, like in sql and other programming languages, allows you to create summaries of data in columns

```
df.groupby(['column you want to group'])['column you want to count'].count()
```

```
df.groupby(['Borough'])['Unique Key'].count()
```

```
Borough
BRONX            10925
BROOKLYN         22247
MANHATTAN        13133
QUEENS           18623
STATEN ISLAND     3848
Unspecified        861
Name: Unique Key, dtype: int64
```

chaining

- in python, multiple operations can be chained together using the dot method

```
df.groupby(['Borough'])['Unique Key'].count().sort_values(ascending=False)
```

```
Borough
BROOKLYN         22247
QUEENS           18623
MANHATTAN        13133
BRONX            10925
STATEN ISLAND     3848
Unspecified        861
Name: Unique Key, dtype: int64
```

filtering columns

to select a column in python, we do the following:	• df[‘column1’]
to select multiple columns, we can do:	• df[[‘column1’, ‘column2’, ‘column3’]]
desired columns, or our subset, can be stored in another dataframe:	• df2 = df[[‘column1’, ‘column2’, ‘column3’]]
we can then use our column subset dataframe, df2, to perform an analysis	

filtering rows

- df[df[‘column name’] == ‘value’]
- df2 = df[df[‘column name’] == ‘value’]
- df3 = df[(df[‘column1’] == ‘value1’) & (df[‘column2’] == ‘value2’)]

filtering rows - exact matching vs. fuzzy matching

- exact matching

```
df[df['Complaint Type'] == 'Noise']
df[df['Complaint Type'] == 'Noise'].count()
```

```
df[df['Complaint Type'].str.contains('Noise')]
```

- fuzzy matching

```
Import pandas as pd
From fuzzywuzzy import process

Data = {
'name' : ['Alice, 'Bob', 'Charlie', 'David'],
'City' : ['New York', 'Los Angeles', 'Miami', 'Chicago']}

Df = pd.DataFrame(data)

# fuzzy matching
```

```
Data = {
'name' : ['Alice, 'Bob', 'Charlie', 'David'],
'City' : ['New York', 'Los Angeles', 'Miami',
'Chicago']}

Df = pd.DataFrame(data)

# Exact matching

Exact_match = df[df['City] == ['New York,
Chicago', 'Chicago']
print("Exact Matching: ")
print(exact_match)
```

Output:
Exact Matching:
0 Alice New York
17 Charlie New York

objectives

- to dig deeper into pandas
- to further understand the nuances of real-world data
- to apply pandas to real-world data

```
x = df.shape[0] # Number of
rows in the dataframe
y = df.shape[1]/2 # Number of
columns divided by 2
z = 'NYC 311 Data' #A string
Variable that represents the
name of the dataset

print(f"x (number of rows):
{x} -> type: {type(x)}]")
print(f"y (half the number of
columns): {y} -> type:
{type(y)}")
print(f"z (string): '{z}' ->
type: {type(z)}")
```

```
data_tuple = (x, y, z)
print("Data tuple:",
data_tuple) #Created a tuple
named data_tuple

data_dict = {
    'rows' : x,
    'half_columns' : y,
    'title' : z
}

print("Data dictionary:",
data_dict)
```

```
complaint_counts =
df.groupby('Complaint Type').size()
print(complaint_counts)
```

```
complaint_counts = df['Complaint
Type'].value_counts()
print(complaint_counts)
```

Group By: Groups the DataFrame by column type and counts the number rows in each grouping.

Value Counts: Directly counts the occurrences of each unique value in the column type, providing a specific, straightforward count.

```
# Chaining operations to filter our
Brooklyn Complaints and select specific
columns from that sub-group
brooklyn_complaints = (df[df['Borough']
== 'BROOKLYN']
                       [['Complaint
Type', 'Resolution Description']])
print(brooklyn_complaints.head())

# Chaining operations to filter our
Queens Complaints and select specific
columns from that sub-group
queens_complaints = (df[df['Borough']
== 'QUEENS']
                       [['Complaint
Type', 'Resolution Description']])
print(queens_complaints)
```

```
filtered_df = df[['Complaint
Type','Borough','Status']]
#Selecting a Specific set of
columns
closed_complaints =
filtered_df[filtered_df['Status']
== 'Closed'] #Selecting rows where
the 'Status' is 'Closed'
df2 = closed_complaints.copy()
print(df2)

noise_complaints =
df2[df2['Complaint Type'] == 'Noise
- Street/Sidewalk']
print(noise_complaints)

#Comparing the noise levels of
environmental noise
```

Adding and Removing Columns

```
# Adding and Removing Columns
df['Created Date'] = pd.to_datetime(df['Created Date'],
errors='coerce')
df['Closed Date'] = pd.to_datetime(df['Closed Date'],
errors='coerce')

# Created 'Request Length' Column from the subtraction of the
created date from the closed date to find the amount of days
df['Request Length'] = (df['Closed Date'] - df['Created
Date']).dt.days

#Remove the 'Request Length' column
df.drop(columns=['Request Length'], inplace=True)

#Rename Columns
df.rename(columns={'Complaint Type':'Type', 'Created Date' :
'Date'}, inplace=True)
print(df)
```

Creating Subsets and Dropping Rows

```
# Create a subset with a specific column name
subset_df = df[['Type', 'Borough', 'Date']]

# Drop a row based on if a 'Borough' is
'Unspecified'
subset_df = subset_df[subset_df['Borough'] !=
'Unspecified']

print(subset_df)
```

Conditional/Logic Tests on Dataframes

```
#Filter for 'Illegal Parking' complaints in
'MANHATTAN'

filtered_subset_df =
subset_df[(subset_df['Type'] == 'Illegal
Parking') & (subset_df['Borough'] ==
'MANHATTAN')]
print(filtered_subset_df.df)
```

Next Week:

- **GitHub Repository**
- **Projects For Practice: Analyzing Data with Python**
- **New Topic: Python for Machine Learning!!! :)**