Welcome to The Carpentries Etherpad!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try https://etherpad.wikimedia.org).

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

---

UCLA Python Intro for Libraries

Workshop details ℹ

Etherpad (this doc): **https://pad.carpentries.org/2024-03-ucla**

Zoom link for both days: https://umn.zoom.us/j/91219326681?pwd=Z0syd2tqcEs2R2Fvd3NZR3dZNDlUUT09

Full curriculum: https://chennesy.github.io/lc-python-intro/

Setup instructions: https://chennesy.github.io/lc-python-intro/#installing-python-using-anaconda

Data: https://chennesy.github.io/lc-python-intro/files/data.zip
Download the **data** folder from the setup page and move it to a **dedicated folder on your Desktop** before the workshop

JupyterHub: https://iassisthub.oarc.ucla.edu
1. Sign in CI Login - big orange button
2. Choose Google option from dropdown
3. Enter email with @g.ucla.edu there
4. Sign in with UCLA login

Post-workshop survey: https://forms.gle/oiQ3RpwPjjPzfm6Y9

☮ **Code of Conduct:** https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

DAY 2

Agenda (Day 2) 🗺

- **Libraries & Pandas**
- **For Loops**
- **Looping Over Data Sets**
- **Using Pandas**
    - Break (10 min)
- **Conditionals**
- **Writing Functions**
- **Tidy Data & Visualizing from Pandas (Tim - new episode) - Mostly all Viz**
- **Post-workshop survey: https://forms.gle/oiQ3RpwPjjPzfm6Y9**

Team & Sign-in

Name | *pronouns* | *Affiliation* | *email*

- Cody Hennesy / he/him / U Minnesota / chennesy@umn.edu
- Geno Sanchez / he|him / UCLA / genosanchez@library.ucla.edu
- Kathy Monahan / she/her / UCLA / kathel13@g.ucla.edu
- Yoonha Hwang / they/them / UCLA / yhwang813@gmail.com
- Kim Mc Nelly / they/them / UCLA / kmcnelly@library.ucla.edu
- Scott Peterson / he him /UCB /speterso@berkeley.edu
- Jillian Wallis / she/they / UCLA/USC / wallisj@usc.edu
- Cristina Berron/she/her/UCLA/berronc@law.ucla.edu
- Maira Hernandez-Andrade/UCLA/mandrade27@ucla.edu
- Rebecca Farmer/ she, her/ UCLA/ rebeccafarmer@yahoo.com
- Robert Johnson/he,him/UCLA/robertjohnson@library.ucla.edu
- Jamie Jamison / she/her / UCLA /jamison@library.ucla.edu
- Laura Dintzis /she / they / UCLA / laura.dintzis@gmail.com
- Tim Dennis / he /him / UCLA / tdennis@library.ucla.edu

NOTES (Day 2)

From feedback from on Day 1:

- Looking for more practical real-world context for how diff Python functions are relevant
- Want to share the goal of the lesson upfront:
    - Use python and pandas to build reproducible workflows for working with library data
    - CSVs representing library usage data from Chicago Public Library
    - Merge data together and run statistics across a multi-year dataset (basic analysis)
    - Tidy (clean) and visualize the data
- Explain dot-notation
- Good pacing vs. too fast
    - Offer more check-ins
- Nice range of help (though Cody was interrupting too much)!

To open Jupyter Lab
Type 'jupyter lab' to launch from the command line (terminal on Mac or Anaconda Prompt on Windows)

Windows: type the Windows key > Anaconda Prompt
Mac: Open your terminal (command + space > type in 'terminal')

- In the terminal, type 'jupyter lab'

Pandas (continued)

load the library as an alias
```
import pandas as pd
```

read your data
```
df = pd.read_csv('data/2011_circ.csv')
```

#rename you dataframe
```
df_2011 = pd.read_csv('data/2011_circ.csv')
```
A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

#head() - view the first 5 rows
```
df_2011.head()
```

#df_2011.info() - summary of df
```
df_2011.info()
```

#columns - list of column names
df_2011.columns

For Loops

#create a list
odds = [1,3,5,7]

print(odds[0], odds[1], odds[2], odds[3])
output: 1 3 5 7

#use for loop to go through the list and print out each element as it encounters it
for num in odds:

- print(num)

output:
1
3
5
7

for loops syntax:
for **variable** in **collection**:

- # **do** things using variable, such as print

#best practice - use more relavant variable names
for kitten in [2,4,6,8]:

- print(kitten)

output:
2
4
6
8

for x in [2,4,6,8]

- print(x)

output:

2
4
6
8

#range - produces a sequence of numbers. Python index values (“up to, but not including”).

```
for number in range(5):
    print(number)
```

output:

0
1
2
3
4

Accumulators - A common loop pattern is to initialize an *accumulator* variable to zero, an empty string, or an empty list before the loop begins. Then the loop updates the accumulator variable with values from a collection.

```
total = 0
for number in range(1,11):
    print(f'number is: {number} total is: {total}')
    total += number
print(f'at the end of the loop, number is: {number} and total is {total}')
```

output:

output:

```
number is: 1 total is: 0
number is: 2 total is: 1
number is: 3 total is: 3
number is: 4 total is: 6
number is: 5 total is: 10
number is: 6 total is: 15
number is: 7 total is: 21
number is: 8 total is: 28
number is: 9 total is: 36
number is: 10 total is: 45
At the end of the loop, number is 10 and total is 55
```

Use a for loop to process files given a list of their names.
loop through the list to read in each CSV file as a dataframe. Let’s print out the maximum values from the ‘ytd’ (year to date) column for each dataframe.

```
for filename in ['data/2011_circ.csv', 'data/2012_circ.csv']:
```

- data = pd.read_csv(filename)
- print(filename, data['ytd'].max())

output:
data/2011_circ.csv 1678047
data/2012_circ.csv 1707032

Use glob to find sets of files whose names match a pattern.
wildcards:

- * will “match zero or more characters”
- ? will “match exactly one character”

import glob # The glob library contains a function also called glob to match file patterns.
print(f"all csv files in data directory: {glob.glob('data/*.csv')}")

output:
all csv files in data directory: ['data/2011_circ.csv', 'data/2016_circ.csv', 'data/2017_circ.csv', 'data/2022_circ.csv', 'data/2018_circ.csv', 'data/2019_circ.csv', 'data/2012_circ.csv', 'data/2013_circ.csv', 'data/2021_circ.csv', 'data/2020_circ.csv', 'data/2015_circ.csv', 'data/2014_circ.csv']

Use glob and for to process batches of files.
```
for csv in glob.glob('data/*.csv'):
  data = pd.read_csv(csv)
  print(csv, data['ytd'].max())
```

output:
data/2011_circ.csv 1678047
data/2016_circ.csv 3478369
data/2017_circ.csv 3623318
data/2022_circ.csv 6336579
data/2018_circ.csv 4006963
data/2019_circ.csv 4821180
data/2012_circ.csv 1707032
data/2013_circ.csv 2069537
data/2021_circ.csv 6629348
data/2020_circ.csv 8222810
data/2015_circ.csv 3195053
data/2014_circ.csv 2792631

```
#sorted lists
for csv in sorted(glob.glob('data/*.csv')):
    data = pd.read_csv(csv)
    print(csv, data['ytd'].max())
```

output:

```
data/2011_circ.csv 1678047
data/2012_circ.csv 1707032
data/2013_circ.csv 2069537
data/2014_circ.csv 2792631
data/2015_circ.csv 3195053
data/2016_circ.csv 3478369
data/2017_circ.csv 3623318
data/2018_circ.csv 4006963
data/2019_circ.csv 4821180
data/2020_circ.csv 8222810
data/2021_circ.csv 6629348
data/2022_circ.csv 6336579
```

Appending dataframes to a list

```
dfs = [] # an empty list to hold all of our dataframes
counter = 1

for csv in sorted(glob.glob('data/*.csv')):
  data = pd.read_csv(csv)
  print(f'{counter} Saving {len(data)} rows from {csv}')
  dfs.append(data)
  counter += 1
```

output:

```
1 Saving 83 rows from data/2011_circ.csv
2 Saving 82 rows from data/2012_circ.csv
3 Saving 83 rows from data/2013_circ.csv
4 Saving 83 rows from data/2014_circ.csv
5 Saving 83 rows from data/2015_circ.csv
6 Saving 83 rows from data/2016_circ.csv
7 Saving 83 rows from data/2017_circ.csv
8 Saving 83 rows from data/2018_circ.csv
9 Saving 84 rows from data/2019_circ.csv
```

```
10 Saving 85 rows from data/2020_circ.csv
11 Saving 85 rows from data/2021_circ.csv
12 Saving 86 rows from data/2022_circ.csv
```

#view 5 rows of the 5th dataframe in your list

```
dfs[5].head()
```

Concatenating dataframes

```
df = pd.concat(dfs, ignore_index = True)
print(f'Number of rows in df: {len(df)}')
```

output:

```
Number of rows in df: 1003
```

#view last few rows in your dataframe

```
df.tail()
```

```
df.tail(10)
```

#slicing

```
df[50:60]
```

#view values of a specific column

```
df['year']
```

#refer to specific column and row

```
df['year'][0]
```

refer to a range of rows in a specific column

```
df['year'][100:103]
```

#what data type?

```
type(df['year'])
```

output:
pandas.core.series.Series

Summary statistics on columns

max()
df['year'].max()
output:
2022

min()
df['year'].min()
2011

#Summarize columns that hold string objects
df['branch']

We can use the .unique() function to output an array (like a list) of all of the unique values in the branch column
df['branch'].unique()

Use .groupby() to analyze subsets of data

df.groupby('branch')['ytd'].sum()

#output
branch
Albany Park 1024714
Altgeld 68358
Archer Heights 803014
Austin 200107
Austin-Irving 1359700
...
West Pullman 295327
West Town 922876
Whitney M. Young, Jr. 259680
Woodson Regional 823793
Wrightwood-Ashburn 302285
Name: ytd, Length: 88, dtype: int64

#Let’s save the output to a new variable and sort

```
circ_by_branch = df.groupby('branch')['ytd'].sum()
circ_by_branch.sort_values(ascending=False).head(10)
```

output:

```
branch
Renewals - Online                 28441560
Renewals - Auto                   21188737
Downloadable Media                15709651
Harold Washington Library Center   7498041
Sulzer Regional                    5089225
Lincoln Belmont                    1850964
Edgewater                          1668693
Logan Square                       1539816
Rogers Park                        1515964
Bucktown-Wicker Park               1456669
Name: ytd, dtype: int64
```

We can pass multiple columns to groupby() to subset the data even further and breakdown the highest usage per year and branch.

```
circ_by_year_branch = df.groupby(['year', 'branch'])
['ytd'].sum().sort_values(ascending=False)
circ_by_year_branch.head(5)
```

output:

```
year  branch
2020  Renewals - Auto     8222810
2021  Renewals - Auto     6629348
2022  Renewals - Auto     6336579
2019  Renewals - Online    4821180
2018  Renewals - Online    4006963
Name: ytd, dtype: int64
```

view first 10 rows

```
circ_by_year_branch.head(10)
```

output:

```
year  branch
2020  Renewals - Auto      8222810
2021  Renewals - Auto      6629348
2022  Renewals - Auto      6336579
2019  Renewals - Online     4821180
2018  Renewals - Online     4006963
2017  Renewals - Online     3623318
2016  Renewals - Online     3478369
2015  Renewals - Online     3195053
2014  Renewals - Online     2792631
2022  Downloadable Media    2663298
Name: ytd, dtype: int64
```

Use .iloc[] and .loc[] to select DataFrame locations.

You can point to specific locations in a DataFrame using two-dimensional numerical indexes with .iloc[].

```
print(f"Branch: {df.iloc[0,0]} \nYTD circ: {df.iloc[0,-2]}")
```

output:
Branch: Albany Park
YTD circ: 120059

print the same values as above, using the column names

```
print(f"Branch: {df.loc[0,'branch']} \nYTD circ: {df.loc[0, 'ytd']}")
```

output:
Branch: Albany Park
YTD circ: 120059

```
print(f"Branch: {df.loc[0, 'branch']} YTD circ: {df.loc[0, 'ytd']}")
```

output:
Branch: Albany Park YTD circ: 120059

Save DataFrames

```
circ_df = circ_by_year_branch.to_frame()
circ_df.head()
```

Save to CSV

```
circ_df.to_csv('high_usage.csv')
```

hit refresh button if file does not automatically appear in your file browser

Save pickle files

```
circ_df.to_pickle('high_usage.pkl')

new_df = pd.read_pickle('high_usage.pkl')
new_df.head()
```

Conditionals

Use if statements to control whether or not a block of code is executed.

```
checkouts = 11
if checkouts > 10.0:
    print(f'{checkouts}, is over the checkout limit.')
```

output:
11, is over the checkout limit.

```
checkouts = 8
if checkouts > 10.0:
    print(f'{checkouts}, is over the checkout limit.')
```

output:
no output

Conditionals are often used inside loops.

```
checkouts = [3, 5, 12, 22, 0]
for checkout in checkouts:
    if checkout > 10.0:
        print(f'{checkout}, is over the checkout limit.')
```

output:
12 is over the checkout limit.
22 is over the checkout limit.

Use else to execute a block of code when an if condition is *not* true.

```
checkouts = [3, 5, 12, 22, 0]
for checkout in checkouts:
    if checkout > 10.0:
        print(f'Warning: {checkout} is over the checkout limit.')
```

```
    else:
        print(f'{checkout} is under the limit.')
```

output:

```
3 is under the limit.
5 is under the limit.
*Warning* 12 is over the checkout limit.
*Warning* 22 is over the checkout limit.
0 is under the limit.
```

Use elif to specify additional tests.

You can use elif (short for “else if”) to provide several alternative choices, each with its own test.

```
checkouts = [3, 5, 10, 22, 0]
for checkout in checkouts:
    if checkout > 10.0:
        print(f'Warning: {checkout} is over the checkout limit.')
    elif checkout == 10:
        print(f'{checkout} is at the exact checkout limit.')
    else:
        print(f'{checkout} is under the limit.')
```

output:

```
3 is under the limit.
5 is under the limit.
10 is at the exact checkout limit.
Warning: 22 is over the checkout limit.
0 is under the limit.
```

You can use conditionals inside of a loop to adjust values as the loop iterates.

```
checkouts = 15.0
for i in range(5): # execute the loop 5 times
    print(f'{i} : {checkouts}')
    if checkouts >= 30.0:
        print('too many checkouts')
        checkouts = checkouts - 5.0
    else:
        print('too few checkouts')
        checkouts = checkouts + 10.0
print(f'final checkouts: {checkouts}')
```

output:

0 : 15.0
too few checkouts
1 : 25.0
too few checkouts
2 : 35.0
too many checkouts
3 : 30.0
too many checkouts
4 : 25.0
too few checkouts
final checkouts: 35.0

Writing Functions

#Define a function using def with a name, parameters, and a block of code.

```
def print_greeting():
    print('Hello!')
```

output:
no output

#you must call the function to execute the code it contains

```
print_greeting()
```

output:
Hello!

Arguments in call are matched to parameters in definition.

```
def print_date(year, month, day):
```

- joined = f'{year}/{month}/{day}'
- print(joined)

```
print_date(1871, 3, 19)
```

output:
1871/3/19

Functions with defined parameters will result in an error if they are called without passing an argument:

Use return to pass values back from a function.

```
def calc_fine(days_overdue):
    if days_overdue <= 10:
        fine =  days_overdue * 0.25
    else:
        fine = (days_overdue * 0.25) + (days_overdue * .50)
    return fine

fine = calc_fine(12)
f'Fine owed: ${fine:.2f}' #specify the number of float decimals to display
```

output:

'Fine owed: $9.00'

Variable Scope

When we define a variable inside of a function in Python, it's known as a local variable, which means that it's not visible to – or known by – the rest of the program. Variables that we define outside of functions are global and are therefore visible throughout the program, including from within other functions. The part of a program in which a variable is visible is called its scope.

```
initial_fine = 0.25
late_fine = 0.50

def calc_fine(days_overdue):
    if days_overdue <= 10:
        days_overdue =  days_overdue * initial_fine
    else:
        days_overdue = (days_overdue * initial_fine) + (days_overdue * late_fine)
    return days_overdue


fine = calc_fine(12)
print(f'Fine owed: ${fine:.2f}')
print(f'Fine rates: ${initial_fine:.2f}, ${late_fine:.2f}')
print(f'Days overdue: {days_overdue}')
```

output:

```
Fine owed: $9.00
Fine rates: $0.25, $0.50
```

NameError

EXERCISES (Day 2) ✏

Exercise 1 ✏

PRACTICE ACCUMULATING

Fill in the blanks to sum the lengths of each string in the list, ["book", "magazine", "dvd"]

```
total = 0
for word in ["book", "magazine", "dvd"] :
    ____ += ___(word)
print(total)
```

```
total = 0
for word in ["book", "magazine", "dvd"] :
```

- total += len(word)

```
print(total)
```

output:
15

Exercise 2 ✏

DETERMINING MATCHES
Which of these files would be matched by the expression glob.glob('data/*circ.csv')?

- A. data/2011_circ.csv
- B. data/2012_circ_stats.csv
- C. circ/2013_circ.csv
- D. Both 1 and 3

answer:

Only item 1 is matched by the wildcard expression 'data/*circ.csv'.
expression starts off with 'data' which invalides C option. The end of the expression ends with 'circ.csv' which invalidates B option. Which leaves A option.

Exercise 3 ✏

MINIMUM CIRCULATION PER YEAR

Modify the following code to print out the lowest value in the ytd column from each year/file.

```
import pandas as pd
for csv in sorted(glob.glob('data/*.csv')):
```

- data = pd.read_csv(____)
- print(csv, data['____'].____())

Exercise 4 ✏

UNIQUE ITEMS

How would you display:

1. All of the unique zip codes in the dataset?
2. The number of unique zip codes in the dataset?

Exercise 5 ✏

DISPLAYING ROWS AND COLUMNS

How would you use slicing and column names to select the following subsets of rows and columns from the circulation DataFrame?

1. The city column.
2. Rows 10 to 20.
3. Rows 20 to 30 from the zip code column.

Exercise 6 ✏

SUMMARY STATISTICS AND GROUPBY()
We can apply mean() to pandas series' in the same way we used sum(), min(), and max() above. How would you display the following?

1. The mean number of ytd checkouts grouped by zip code?
2. The mean number of ytd checkouts grouped by zip code, and sorted from smallest to largest?

Exercise 7 ✏

ENCAPSULATION

Fill in the blanks to create a function that takes a single filename as an argument, loads the data in the file named by the argument, and returns the minimum value in that data.

```
import pandas

def min_in_data(____):
    data = ____
    return ____
```

Tidy/DataViz section

Tidy data is a standard way of organizing data values within a dataset, making it easier to work with. Here are the key principles of tidy data:
1. Every column holds a single variable, like "month" or "temperature."
2. Every row represents a single observation, like circulation counts by branch and month.
3. Every cell contains a single value

The blow visual might help orient us.

- Tidy Data <https://d33wubrfki0l68.cloudfront.net/6f1ddb544fc5c69a2478e444ab8112fb0eea23f8/91adc/images/tidy-1.png>
- See this chapter on R for Data Science https://r4ds.had.co.nz/tidy-data.html#tidy-data
    - Hadley Wickham coined Tidy Data in the R community and it has been picked up by other data communities (Tableau, Python, other stats programs)

Benefits of Tidy Data

Transforming our data into a tidy data format advantages:
- Python tools, such as visualization, filtering, and statistical analysis libraries, work better with data in a tidy format.
- Tidy data makes transforming, summarizing, and visualizing information easier. For instance, comparing monthly trends or calculating annual averages becomes more straightforward.
- As datasets grow, tidy data ensures that they remain manageable and analyses remain accurate.

Python has a reshaping function called 'melt' and we can melt our data down (so month columns become

a variable called 'month')

```
df_long = df.melt(id_vars=['branch', 'address', 'city', 'zip code', 'ytd', 'year'],
                  value_vars=['january', 'february', 'march', 'april', 'may', 'june',
                              'july', 'august', 'september', 'october', 'november', 'december'],
                  var_name='month', value_name='circulation_count')
```

Some categories we don't need for our analysis, so let's drop them:

```
df_long = df_long[~df_long['branch'].isin(["Renewals - Online", "Renewals - Auto", 'Renewals - Itivia',
    'Renewals - Phone', 'Talking Books and Braille'])]
```

DataViz

We've prepared a tidy data set and we can read it in directly from Cody's github repo.

Read in reshaped data:

```
df_long = pd.read_csv('https://raw.githubusercontent.com/chennesy/lc-python-intro/main/episodes/data/circ_long.tsv', delimiter='\t', index_col=-1, parse_dates=True)  # for tab-delimited
```

1. We can read in by URL to CSV or TSV or other formats (
2. We can tell Pandas how what we are reading in is delimited
3. Setting index col to the end column (a date) column enables time-series analysis and data handling and visualization, among other things (|, ;, etc.)

Subset by one branch:

```
albany = df_long[df_long['branch'] == 'Albany Park']
albany = albany.sort_values(by='date')
albany['circulation_count'].plot()


# the main, oldest ploting package in python
import matplotlib.pyplot as plt

albany['circulation_count'].plot(title='Circulation Count Over Time', figsize=(10, 5), color='blue')
# Adding labels and showing the plot
plt.xlabel('Date')
plt.ylabel('Circulation Count')
plt.show()
```

A different treatment using area plot:

```
albany['circulation_count'].plot(kind='area', title='Circulation Count Area Plot at Albany Park', alpha=0.5)
plt.xlabel('Date')
plt.ylabel('Circulation Count')
plt.show()
```

A histogram:

```
albany['circulation_count'].plot(kind='hist', bins=20, title='Distribution of
Circulation Counts at Albany Park')
plt.xlabel('Circulation Count')
plt.show()
```

How do you think you would show a boxplot?

Let's switch back to the full dataset in `df_long` and use a dedicated plotting package in Python called plotly. First let's install the package.

```
import plotly.express as px
```

```
# Creating a line plot for a few selected branches to avoid clutter
selected_branches = df_long[df_long['branch'].isin(['Altgeld',
'Archer Heights',
'Austin',
'Austin-Irving',
'Avalon'])]
fig = px.line(selected_branches, x=selected_branches.index, y='circulation_count',
color='branch', title='Circulation Over Time for Selected Branches')
fig.show()
```

```
# Aggregate circulation by branch
total_circulation_by_branch = df_long.groupby('branch')
['circulation_count'].sum().reset_index()

# Create a bar plot
fig = px.bar(total_circulation_by_branch, x='branch', y='circulation_count',
title='Total Circulation by Branch')
fig.show()
```

Agenda (Day 1)

- **Introduction**
- **Getting Started**
- **Variables and Types**
- **BREAK (10 min)**
- **Lists**
- **Built-in Functions & Help**
- **Libraries & Pandas**

Python Package Index: https://pypi.org/

- **Pandas** (https://pandas.pydata.org/)- tabular data analysis tool.
- **Pymarc** (https://pypi.org/project/pymarc/) - for working with bibliographic data encoded in MARC21.
- **Matplotlib** (https://matplotlib.org/) - data visualization tools.
- **BeautifulSoup** (https://pypi.org/project/beautifulsoup4/) - for parsing HTML and XML documents.
- **Requests** (https://pypi.org/project/requests/) - for making HTTP requests (e.g., for web scraping, using APIs)
- **Scikit-learn** (https://scikit-learn.org/stable/) - machine learning tools for predictive data analysis.
- **NumPy** (https://numpy.org/) - numerical computing tools such as mathematical functions and random number generators.

Team & Sign-in

Name* | *pronouns* | *Affiliation* | *email
Geno Sanchez ☆ /he, him/UCLA/genosanchez@library.ucla.edu
Scott Peteson /he, him/UCB/speterso@berkeley.edu
Jamie Jamison / she, her / UCLA / jamison@library.ucla.edu
Cody Hennesy ☆ / he/him / U Minnesota / chennesy@umn.edu
Laura Dintzis /she/they/UCLA/laura.dintzis@gmail.com
Kathy Monahan / she/her/UCLA/kathel13@g.ucla.edu
Robert Johnson/he,him/UCLA/robertjohnson@library.ucla.edu
Rebecca Farmer / she, her/UCLA/rebeccafarmer@yahoo.com
Cristina Berron/she/her/UCLA/berroncristina@gmail.com
Yoonha Hwang / they,them / UCLA / yhwang813@gmail.com
Jillian Wallis / she/they / UCLA/USC / jwallisi@g.ucla.edu/wallisj@usc.edu
Katherine Ramirez / she, her / Stanford University / ramzkatherine@gmail.com

Maira Hernandez-Andrde/she, her/UCLA/mandrade27@ucla.edu
Katy Egts/she, her/Washington University in St. Louis/egts@wustl.edu
Dorothy Alvarado/she/her/UCLA/dorothyalva08@g.ucla.edu
Mohsin Ali /he/him/UCLA/mohsinmalikali@ucla.edu
Kim Mc Nelly / they, them / UCLA / kmcnelly@library.ucla.edu
Paige Wilcox / she,her / Stanford / pawilcox@stanford.edu
Benjamin Manuel / he/they / UCLA / bmanuel@library.ucla.edu
Nat Stewart // they/them // UCLA // natstewboy@g.ucla.edu
Leigh Phan / she/her / UCLA / leighphan@ucla.edu
Sarah T. Roberts/ she,her/ UCLA/sarah.roberts@ucla.edu

Notes

To open Jupyter Lab
Type 'jupyter lab' to launch from the command line (terminal on Mac or Anaconda Prompt on Windows)

Windows: type the Windows key > Anaconda Prompt
Mac: Open your terminal (command + space > type in 'terminal')

- In the terminal, type 'jupyter lab'

If you are encountering any issues with anaconda, try:
JupyterHub: https://iassisthub.oarc.ucla.edu
*** Backup web option if you have issues installing or running Python/Jupyter on your own machine***
***Use g.ucla.edu account to login**

Setting up

Start a new project folder, naming it **'lc-python'** (this will be our working directory)
Within your working directory, create a folder called '**data**'
Create a new notebook within the working directory

In Jupyter Notebooks, each line is called a cell. Cells can include different types of information: e.g. Code, Markdown

Markdown
- can be used to format text such as documentation to go along w/ your code
- You can create Headers by using the # symbol before text

- # Header 1
- ## Header 2
- ### Header 3, etc.
- asterisks or '-' to create bullet points
- 1. Create a numbered list
- 2.
- 3.

Switch the text type of a cell from 'Markdown' to 'Code' to enter code

- M make the active cell a Markdown cell
- Y make the active cell a code cell
- reference: https://gist.github.com/discdiver/9e00618756d120a8c9fa344ac1c375ac

Variables

```
age = 42
name = 'Ahmed'
```

formatted strings (aka 'f-strings')

```
print(f'{name} is {age} years old')

age = age + 3
print(f'Age equals: {age}')
```

Types

Find the type of a value by preceding the value with 'type()' and inputting the value in the parentheses.

```
type(140.2)
type(age)
type(name)
type(print)
```

The terms function and method can be used interchangeably.

Indexing

Allows us to access elements within a list. In Python the index always begins at 0 (not 1).
Format example: variable[]

```
library = 'Alexandria'
library[0]

name[2]
```

whos: to get a list of all the variables you have created in your Jupyter notebook, can type in 'whos' the whos magic command is something unique to Jupyter Notebooks

Slicing

```
[start:stop]
```

```
library
library[0:3]   #counts the first letter, up to but not including the 3rd
library[2:5]
```

Length -- len()

```
len('Babel')
len(library)
```

len() can be helpful when you're working with data from an API or online that you are less familiar with; you can get more information about the data

```
age = 42
older_age = age + 3
age = 50

print(f'Older age is {older_age} and age is {age}')
```

Lists

1. Lists are mutable.
2. Lists are also hetereogeneous
each value needs to be separated by commas

```
zip_codes = [60625, 60827, 60632, 60644, 60634]
print(f'Zip codes: {zip_codes}')
```

Python built-in functions: https://docs.python.org/3/library/functions.html

---

Python for Libraries (Day 1) Exercises

Exercise 1 ✏

SWAPPING VALUES

```
x = 1.0
y = 3.0
swap = x
x = y
y = swap
```

Draw a table showing the values of the variables in this program after each statement is executed.

In simple terms, what do the last three lines of this program do?

```
swap = x  #  x = 1.0 y = 3.0 swap = 1.0
x = y     #  x = 3.0 y = 3.0 swap = 1.0
y = swap  #  x = 3.0 y = 1.0 swap = 1.0
```

Exercise 2 ✏

PREDICTING VALUES

```
initial = "left"
position = initial
initial = "right"
```

What is the final value of **position**? (Try to predict the value without running the program, then check your prediction.)

Exercise 3 ✏

CAN YOU SLICE INTEGERS?

```
a = 123
```

What happens if you try to get the second digit of **a**?

```
a[2]
# If we add quotes to the value (a = '123') then the value becomes a string. However, it will no longer be treated as a numeric value.
```

Exercise 4 ✏

SLICING

```
library_name = 'Library of Babel'

print(f'library_name[1:3] is: {library_name[1:3]}')
```

- Try the following to figure out how these different forms of slicing work
- Replace **low** and **high** with index positions of your choosing

1. What does library_name[low:] (without a value after the colon) do?
2. What does library_name[:high] (without a value before the colon) do?
3. What does library_name[:] (just a colon) do?
4. What does library_name[number : negative-number] do?

Exercise 5 ✏

FILL IN THE BLANKS

Fill in the blanks so that the program below produces the output shown.

```
values = __
values._____(1)
values._____(3)
values._____(5)
print('first time:', values)
values = values[_:_]
print('second time:', values)
```

expected output:

```
first time: [1, 3, 5]
second time: [3, 5]
```

Exercise 6 ✏

FROM STRINGS TO LISTS AND BACK

```
print('string to list:', list('book'))
print('list to string:', ''.join(['a', 'r', 't', 'i', 'c', 'l', 'e']))
```

expected output:

```
['b', 'o', 'o', 'k']
'article'
```

1. Explain in simple terms what **list**('book') does.
2. What does '-'.**join**(['x', 'y']) generate? Note that the first set of single quotes is not empty.

Exercise 7 ✏

WORKING WITH THE END

```
resources = ['books','periodicals','DVDs','maps','databases','abstracts']

print(resources[-1])
```

1. How does Python interpret a **negative** index?
2. If a list or string has **N** elements, what is the **most negative** index that can safely be used with it, and what location does that index represent?
3. If **resources** is a list, what does **del resources[-1]** do?
4. How can you display all elements but the last one without changing **resources**? (Hint: you will need to combine slicing and negative indexing.)

Exercise 8 ✏

SPOT THE DIFFERENCE

1. Predict what each of the print statements in the program will print.
2. Does **max(len(cataloger), assistant_librarian)** run or produce an error message? If it runs, does its result make any sense?

```
cataloger = "metadata_curation"
assistant_librarian = "archives"

print(max(cataloger, assistant_librarian))
print(max(len(cataloger), assistant_librarian))
```
