Lecture 4:

- Python Basics continued
- Jupyter notebooks

Announcements:

- HW-1 out, due next Thursday
- No class on Tuesday (MHS)
- Quiz 2 next Thursday (over HW-1, Python basics)
Python For Loops

For loops take the general form:

```
for var in iterator:
    statement 1
    statement 2
    ...
```

where an iterator is an object that can be “iterated over”

For example, for loop over lists (and other collection types):

```python
x = [10, 20, 30, 40]
for x in x:
    print(x)
```

Can also use the `range(start, stop, step)` function:

- returns an iterator from `start` up to not including `stop`
- only the `start` is required ...

```python
x = [10, 20, 30, 40, 50, 60]
for i in range(0, len(x), 1):
    # prints each element
    print(x[i])

for i in range(len(x)):
    # same as above
    print(x[i])

for i in range(1, 4):
    # prints 20, 30, 40
    print(x[i])

for i in range(0, 5, 2):
    # prints 10, 30, 50
    print(x[i])

for i in range(-1, -7, -1):
    # prints list in reverse
    print(x[i])

for i in range(5, 1, -2):
    # prints 60, 40
    print(x[i])
```
List Comprehensions

List comprehensions provide a simple “list builder” notation of the form:

\[
[expr \ for \ var \ in \ iterator]
\]

Examples:

\[
xs = [10, 20, 30, 40, 50, 60]
\]

\[
ys = [v * 2 \ for \ v \ in \ xs] \quad \# \ [20, 40, 60, 80, 100, 120]
\]

\[
ys = [2 ** i \ for \ i \ in \ range(5)] \quad \# \ [1, 2, 4, 8, 16]
\]

\[
ys = [[x, y] \ for \ x \ in \ [1,2] \ for \ y \ in \ [3,4]] \quad \# \ [[1,3],[1,4],[2,3],[2,4]]
\]

\[
ys = [x * y \ for \ x \ in \ [1,2] \ for \ y \ in \ [3,4]] \quad \# \ [3, 4, 6, 8]
\]

Python provides similar capability for sets and dictionaries:

\[
>>> \{x % 2 \ for \ x \ in \ range(5)} \quad \# \ is \ the \ modulo \ operator
\{0, 1\}
\]

\[
>>> \{k : True \ for \ k \ in \ ['a','b']\}
\{'a': True, 'b': True\}
\]
Basic Python Exception Handling

Exception handling in Python is similar to other languages

- exceptions can be caught ("handled") using **try**-**except** blocks
- Python also provides a **with** clause, which we’ll discuss later

Basic form of a **try**-**except**:

```python
try:
    statement 1
...
except:
    statement 1
...
```

Simple examples:

```python
try:
    xs = [10]
    print(xs[1])
except:
    print('Oops ')

v = None
try:
    v = int(some_string)
except:
    print(f'"{some_string}" not a number')
```
Can also handle different types of exceptions:

```python
try:
    statements
except ExceptionType1 as e1:
    statements
except ExceptionType2 as e2:
    statements
```

For example:
```
v = None
xs = ['a']

# this prints: "bad index: list index out of range"
try:
    v = int(xs[1])
except IndexError as e1:
    print('bad index:', e1)
except ValueError as e2:
    print('bad value:', e2)

# this prints: "bad value: invalid literal for int() ...
try:
    v = int(xs[0])
except IndexError as e1:
    print('bad index:', e1)
except ValueError as e2:
    print('bad value:', e2)
```

The **finally** clause is also supported...try-except-finally

- **finally** is called after try with or without an exception
- useful for closing resources, cleaning up before moving on
- called even if an **except** clause returns or exits
**Jupyter Notebooks**

Jupyter supports web-based “interactive” notebooks:

- A mix of formatted text and code areas (called **cells**)
- Code can be edited and executed, with results displayed
- Useful for creating “executable” reports and documentation
- Also useful for documenting how to use libraries, etc.

(*) Note Jupyter used to be called iPython, so file names end in **ipynb**

To get started with notebooks from the command line run:

```
jupyter notebook
```

Starts server and opens browser page:

(*) There is also integrated support in VSCode
From the home page, you can:

- create new notebooks
- create new files and edit them
- create a new terminal
- manage open terminals and notebooks
- open existing notebooks or files

Note that we will primarily be creating notebooks in this class.

After creating a new notebook you should see something like the following:

- there is one cell in the notebook (ready to be edited)
- additional menu options, etc.
There are a number of shortcut keys, to see them type "H"

- this is the only shortcut key you absolutely need to remember!

There are two fundamental types of cells:

1. Code cells, where you can write python code
2. Markdown code for text

Text cells use "markdown" format

- *this is bold* for this is bold
- **this is italics** for this is italics
- # Level 1 heading and ## Level 2 heading
- and so on ...
Code cells contain python code fragments

- the code cells share state (same execution environment)
- so variables, functions, libraries, etc. from one cell available in later cells

Each code cell can be run and rerun individually

All of the code cells in a notebook can also be run sequentially

- For example, under Cell → Run All
- Can also restart Kernel, e.g., under Kernel → Restart & Run All
- There are various options ...

To stop the server

- Click the Quit button in the home page
- Or Ctrl-c in the terminal window (and type Y at the prompt)
For HW-1, your job is to fill out `hw1_examples.ipynb`:

You must:

- Replace “Your name here” with your name
- Add sufficient tests (cells) for each function to ensure they work
- Add text to last section on challenges, issues, observations

We will use notebooks for documenting our code throughout the course