# Lecture-Test-student-solutions

August 3, 2022

## 1 PRG550 Lecture Test Solutions

July 19, 2022 July 20, 2022

There are 12 questions with 120 total available points. - The test will be graded out of 100. - If you score more than 100, the additional points will count as bonus. - Partial marks will be allocated to multi-part questions

Skim ahead and read all questions before starting.

Work on the ones you are most confident with first

---

1. Provide code to answer the below questions in the indicated cells
2. Run the cell below to import the needed libraries
3. Save your notebook often
4. Submit your notebook online with Lynx text-based web browser Course Grader - Online Submission Site

Some helpful links:

Pandas User Guide

https://pandas.pydata.org/docs/getting_started/intro_tutorials

Using Course Grader on Raspberry Pi with Lynx

[2]:
```
# run this cell to import libraries
import pandas as pd
import numpy as np
from functools import reduce
import json
import string
import re
```

```
import yfinance as yf
```

## 2 Loops - Left/Right Arrows

Write code to print left and right pointing arrows using one of the below single ascii characters

```
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

for example:

```
arrow size: 3
   *
  **
 ***
  **
   *
arrow size: 5
j
jj
jjj
jjjj
jjjjj
jjjj
jjj
jj
j
arrow size: 4
 -------
  -----
   ---
    -
arrow size: 2
 ,
,,,
```

[ ]:
```
def down_arrow(char, arrow_size):
    # your code here
def up_arrow(char, arrow_size):
    # your code here
```

```
def left_arrow(char, arrow_size):
    # your code here
def right_arrow(char, arrow_size):
    # your code here
```

### 2.1 Evaluate Right/Left Arrows

[ ]:
```
def right_arrow(char,arrow_size):
    for r in range(1,arrow_size):
        print(char*r)
    for r in range(arrow_size,0, -1):
        print(char*r)

def left_arrow(char,arrow_size):
    for r in range(1,arrow_size):
        print(' '*(arrow_size-r+1), end="")
        print(char*r)
    for r in range(arrow_size,0, -1):
        print(' '*(arrow_size-r+1), end="")
        print(char*r)

def up_arrow(char, arrow_size):
    for r in range(1, arrow_size+1):
        print(' '*(arrow_size-r), end="")
        print(char*(2*r-1))

def down_arrow(char, arrow_size):
    for r in range(arrow_size, 0, -1):
        print(' '*(arrow_size-r), end="")
        print(char*(2*r-1))
```

[ ]:
```
ascii_characters = string.digits+string.ascii_letters+string.punctuation

arrow_char_idx = np.random.randint(0, len(ascii_characters))
arrow_char = ascii_characters[arrow_char_idx]
arrow_size = np.random.randint(2, 10)
print(f"arrow size: {arrow_size}")
left_arrow(arrow_char,arrow_size)
```

```
arrow_char_idx = np.random.randint(0, len(ascii_characters))
arrow_char = ascii_characters[arrow_char_idx]
arrow_size = np.random.randint(2, 10)
print(f"arrow size: {arrow_size}")
right_arrow(arrow_char,arrow_size)

arrow_char_idx = np.random.randint(0, len(ascii_characters))
arrow_char = ascii_characters[arrow_char_idx]
arrow_size = np.random.randint(2, 10)
print(f"arrow size: {arrow_size}")
down_arrow(arrow_char,arrow_size)

arrow_char_idx = np.random.randint(0, len(ascii_characters))
arrow_char = ascii_characters[arrow_char_idx]
arrow_size = np.random.randint(2, 10)
print(f"arrow size: {arrow_size}")
up_arrow(arrow_char,arrow_size)
```

# 3 Lists-1

Write a Python program to remove ALL duplicates from a list. Find the list item that has the highest frequency For example, the list [2, 3, 10, 10, 8, 9, 2, 8, 16, 3, 2, 3] would result in a list: [2, 3, 10, 8, 9, 16]

[ ]:
```
def remove_duplicates(input_list):
    # your code here

    return result_list

print(remove_duplicates(input_list))
```

## 3.1 Evaluate Lists-1

[ ]:
```
def remove_duplicates(input_list):
    unique_num = {}
    for num in input_list:
        if unique_num.get(num) is None:
```

```
            unique_num[num] = 1
    return unique_num.keys()
```

[ ]:
```
remove_duplicates([2, 3, 10, 10, 8, 9, 2, 8, 16, 3, 2, 3])
```

# 4 Lists-2

Given the 2 dimensional list `letters`

```
letters = [
 ['a', 'b', 'c', 'd'],
 ['e', 'f', 'g', 'h'],
 ['i', 'j', 'k', 'l'],
 ['m', 'n', 'o', 'p']
 ]
```

use indexing to produce: 1. `['h', 'g', 'f']` 1. `['n', 'o', 'p']` 1. `['k', 'l']`

[ ]:
```
# your code here
```

## 4.1 Evaluate Lists-2

[ ]:
```
letters[1][3:0:-1], letters[3][1:], letters[2][2:]
```

[ ]:
```

```

# 5 Dictionary-1

Given a class of students and the courses they are taking this semester:

```
student_name = ['Jane', 'Hamed', 'Maryam', 'William']
course_list = ['programming', 'calculus', 'physics']
```

Create a dictionary to show the courses each student is taking and their mark. Use `np.random.randint()` from the `numpy` module to randomly assign grades between 45% to 100% for each course taken by the student

```
course_grade = np.random.randint(45,100) # generate random number between 45 and 100
```

For example:

```
class_courses_and_grades = {
    "Jane" : { "courses": {
                    "programming": 70,
                    "calculus": 80,
                    "physics": 85
                            }
            },
    "Hamed" : { "courses": {
                    "programming": 72,
                    "calculus": 76,
                    "physics": 65
                            }
            },
    "Maryam" : { "courses": {
                    "programming": 86,
                    "calculus": 55,
                    "physics": 67
                            }
            },
    "William" : { "courses": {
                    "programming": 56,
                    "calculus": 85,
                    "physics": 68
                            }
            }
    }
```

[ ]:
```
# variables set up

student_name = ['Jane', 'Hamed', 'Maryam', 'William']
course_list = ['programming', 'calculus', 'physics']
```

[ ]:
```
def create_student_grades():
    # your code here
    return student_courses
```

[ ]:
```
print(create_student_grades())
```

### 5.1 Evaluate Dictionary-1

[ ]:
```
def create_student_grades():
    student_courses = {}
    for student in student_name:
        course_grade_list = {}
        for course in course_list:
            course_grade = np.random.randint(45,100)
            course_grade_list[course] = course_grade
        student_courses[student] = course_grade_list
    return student_courses
```

[ ]:
```
create_student_grades()
```

## 6 Dictionary-2

Given a list of ICAO airport codes, and corresponding city names, create a dictionary that maps ICAO:city name

```
ICAO = ["yyz", "nrt", "jfk", "cdg", "sfo"]
city = ["toronto", "tokyo","new york","paris" "san francisco"]
population = [3000000, 14000000, 8400000, 2100000, 875000]
```

Example:

```
airport_codes = { "yyz" : ["toronto", 3000000], "nrt" : ["tokyo", 14000000], "jfk" : ["new york", 8400000],
                  "cdg" : ["paris", 2100000], "sfo" : ["san francisco",875000] }
```

[ ]:
```
# set up variables
ICAO = ["yyz", "nrt", "jfk", "cdg", "sfo"]
city = ["toronto", "tokyo","new york","paris", "san francisco"]
population = [3000000, 14000000, 8400000, 2100000, 875000]
```

[ ]:
```
def create_airport_mapping(icao_list, city_list, population):
    # your code here

    # dict_mapping = ..,?
    return dict_mapping
```

[ ]:
```
print(create_airport_mapping(ICAO, city, population))
```

### 6.1 Evaluate Dictionary-2

[ ]:
```
def create_airport_mapping(icao_list, city_list, population):
    city_population = [[c,p] for c, p in zip(city_list, population)]
    dict_mapping = {i:cp for i, cp in zip(icao_list, city_population)}
    return dict_mapping

create_airport_mapping(ICAO, city, population)
```

[ ]:
```
m = map(lambda i,c,p: (i,[c,p]), ICAO, city, population)
dict(m)
```

## 7 Regex-1

Use regular expressions to replace every occurrence of the character `s` (upper or lowercase), in the string `She sells sea shells by the sea shore.`, with the character `T`.

Hint: https://docs.python.org/3/library/re.html https://docs.python.org/3/howto/regex.html

[ ]:
```
# variable setup
input_string = 'She sells sea shells by the sea shore.'
```

[ ]:
```
# your code here
```

### 7.1 Evaluate Regex-1

[ ]:
```
re.sub('s', 'T', input_string, flags=re.I)
```

## 8 Regex-2

Use regular expressions to extract and print the area code of a list of telephone numbers

`phone_list = ['416-133-8942', '905-312-0123', '647-264-0872']`

example result:

```
416
905
```

647

[ ]:
```
# variable setup
phone_list = ['416-133-8942', '905-312-0123', '647-264-0872']
```

[ ]:
```
# your code here
```

### 8.1 Evaluate Regex-2

[ ]:
```
phone_list = ['888-264-0872', '555-133-8942', '905-312-0123', '123-264-0872']
pattern = r'^\d{3}' # match 3 digits at beginning of string

for ph in phone_list:
    m = re.match(pattern, ph)
    print(m.group(0))
```

## 9 Loops - Armstrong Numbers

Write the code for a Python program that determines how many numbers from 10 to 999 inclusive are Armstrong numbers! A number is an Armstrong number if it is equal to the sum of its own digits raised to the power of the number of digits it contains. So, for example, the number 153 is an Armstrong number because 1^3 + 5^3 + 3^3 = 153

[ ]:
```
def calc_armstrong_numbers(lower_bound, upper_bound):
    # your code here
```

[ ]:
```
calc_armstrong_numbers(10, 1000)
```

### 9.1 Evaluate Armstrong Numbers

[2]:
```
def calc_armstrong_numbers(lower_bound, upper_bound):
    armstrong_numbers = []
    for number_idx in range(lower_bound, upper_bound+1):
        num_str = str(number_idx)
        num_length = len(num_str)

        digit_sum = 0
        for digit in num_str:
```

```
            digit_sum += int(digit) ** num_length
        if digit_sum==number_idx:
            armstrong_numbers.append(number_idx)
    return armstrong_numbers
```

[3]:
```
calc_armstrong_numbers(10, 1000)
```

[3]:
```
[153, 370, 371, 407]
```

## 10 Lambda-1

Given a `integer_list`, use `reduce()` to calculate the sum of the integers

Create a pandas dataframe from `integer_list` and name the column `x` Calculate the total sum of the integers

[ ]:
```
integer_list = list(range(0,20))

# your code here

print(f"Sum using reduce(): {sum_from_reduce}")
print(f"Sum using dataframe: {sum_from_dataframe}")
```

## 11 Evaluate Lambda-1

[9]:
```
from functools import reduce
integer_list = list(range(0,20))

sum_from_reduce = reduce(lambda x,y: x+y, integer_list)

df = pd.DataFrame(integer_list, columns=['x'])
sum_from_dataframe = df['x'].sum()
print(sum_from_reduce, sum_from_dataframe)
assert sum_from_reduce==sum_from_dataframe # this should succeed with no error
```

```
190 190
```

## 12 Lambda-2

Given three integer lists `a_list`, `b_list`, `c_list` use `map()` to create `result_list` where each element is a result of string concatenation from elements in the three input lists. Example:

```
a_list = [1, 2, 3, 4]
b_list = [7, 8, 9, 0]
c_list = [9, 1, 8, 6]

result_list = ['179', '281', '398', '406']
```

Create a Pandas dataframe from `a_list`, `b_list`, `c_list` and name the columns `x`, `y`, `z` Create a new column called `results` that is a string concatenation for each row of `x`, `y`, `z`

[104]:
```
# run cell to setup variables
list_length = np.random.randint(3,6)
a_list = np.random.randint(0,10, size=list_length)
b_list = np.random.randint(0,10, size=list_length)
c_list = np.random.randint(0,10, size=list_length)
print(f"{list_length}\n{a_list}\n{b_list}\n{c_list}")
```

```
4
[4 4 8 2]
[1 0 4 1]
[4 4 5 0]
```

[ ]:
```
def string_concatenate(list1, list2, list3):
    # your code here
    # result_list = ...
    return result_list

def pandas_string_concatenate(list1, list2, list3):
    # your code here

    return result_list
```

[ ]:
```
print(string_concatenate(a_list, b_list, c_list))
print(pandas_string_concatenate(a_list, b_list, c_list))
```

### 12.1 Evaluate Lambda-2

[20]:
```
list_length = np.random.randint(3,6)
a_list = np.random.randint(0,10, size=list_length)
b_list = np.random.randint(0,10, size=list_length)
c_list = np.random.randint(0,10, size=list_length)

print(f"{list_length}\n{a_list}\n{b_list}\n{c_list}")

def string_concatenate(list1, list2, list3):
    result_list = map(lambda x,y,z: str(x)+str(y)+str(z), a_list, b_list, c_list)
    return list(result_list)

def pandas_string_concatenate(list1, list2, list3):
    df = pd.DataFrame({'x':list1, 'y':list2, 'z':list3})
    df['result'] = df.apply(lambda r: reduce(lambda x,y: str(x)+str(y), r), axis=1) # reduce() for repeated concatenation
    result_list = df['result'].values.tolist()
    return result_list

def pandas_string_concatenate2(list1, list2, list3):
    df = pd.DataFrame({'x':list1, 'y':list2, 'z':list3}, dtype='str') # cast integer in lists to string type
    df['result'] = df.apply(lambda r: reduce(lambda x,y: x+y, r), axis=1) # reduce() for repeated concatenation
    df['result2'] = df['x'] + df['y'] + df['z'] # string concat
    result_list = df['result2'].values.tolist()
    return result_list

print(string_concatenate(a_list, b_list, c_list))
print(pandas_string_concatenate(a_list, b_list, c_list))
print(pandas_string_concatenate2(a_list, b_list, c_list))
```

```
3
[7 3 5]
[3 1 9]
[0 3 1]
['730', '313', '591']
['730', '313', '591']
['730', '313', '591']
```

# 13 Lambda-3

Given a list of integers `random_integers` use `filter()` to remove all values less than `filter_value`

Create a dataframe with `random_integers` as a column named `x` Filter the dataframe to only display values larger than `filter_value`

[113]:
```
filter_value = np.random.randint(0,100)
input_list = np.random.randint(0,100, size=30)

print(f"{filter_value}, {input_list}")
print(input_list)
```

```
24, [76 88 94 85 28 49 55 93 79 38 28 88 49  7 27 84 74 75 75 41 69 65 27 10
 47 36 99 47 85 77]
[76 88 94 85 28 49 55 93 79 38 28 88 49  7 27 84 74 75 75 41 69 65 27 10
 47 36 99 47 85 77]
```

[ ]:
```
def filter_list(threshold, input_list):
    # your code here
    # result_list = ...

    return result_list
```

[ ]:
```
print(list(filter_list(filter_value, random_integers)))
```

[ ]:
```
def filter_list_dataframe(threshold, input_list):
    # your code here

    return result_list
```

[ ]:
```
print(list(filter_list_dataframe(filter_value, random_integers)))
```

## 13.1 Evaluate Lambda-3

[115]:
```
def filter_list(threshold, input_list):

    filtered_list = filter(lambda v: v>threshold, input_list)
    return filtered_list
print(list(filter_list(filter_value, input_list)))
```

```
[76, 88, 94, 85, 28, 49, 55, 93, 79, 38, 28, 88, 49, 27, 84, 74, 75, 75, 41, 69,
65, 27, 47, 36, 99, 47, 85, 77]
```

[117]:
```python
def filter_list_dataframe(threshold, input_list):
    df = pd.DataFrame(input_list, columns=['x'])
    df_new = df.loc[df['x']>filter_value].copy()
    return df_new['x'].values

print(filter_list_dataframe(filter_value, input_list))
```

```
[76 88 94 85 28 49 55 93 79 38 28 88 49 27 84 74 75 75 41 69 65 27 47 36
 99 47 85 77]
```

## 14 Lambda-4

Given the following list of 3 records (each containing 4 fields):

```
item:         book_name:                      qty:      price:
34587         Learning Python, Mark Lutz      4         40.95
98762         Programming Python, Mark Lutz   5         56.80
77226         Head First Python, Paul Barry   3         32.95
```

Write a Python program, which returns a list of tuple containing `(book_name, price)`. The Python program must use `lambda` and `map`.

ex:

```
[('Learning Python, Mark Lutz', '40.95'), ('Programming Python, Mark Lutz', '56.80'), ('Head First Python, Paul Barry', '32.95')]
```

[118]:
```python
# variable setup
book_records = [
    ['34587','Learning Python, Mark Lutz','4','40.95'],
    ['98762','Programming Python, Mark Lutz','5','56.80'],
    ['77226','Head First Python, Paul Barry','3','32.95'],
]
```

[ ]:
```python
def create_tuple(book_records):
    # your code here

    return list(map_result)
```

[ ]:
```
result = create_tuple(book_records)
print(result)
```

### 14.1 Evaluate Lambda-4

[119]:
```
def create_tuple(book_records):
    map_result = map(lambda row: (row[1],row[3]), book_records)
    return(list(map_result))
```

[120]:
```
result = create_tuple(book_records)
print(result)
```

```
[('Learning Python, Mark Lutz', '40.95'), ('Programming Python, Mark Lutz',
'56.80'), ('Head First Python, Paul Barry', '32.95')]
```

## 15 Functions-1

Write a function that prints the number of positional arguments, keyword arguments that was provided to it

Example output

```
# of positional parameters: 3
# of keyword parameters: 2
```

[4]:
```
# set up variables
a=4
b=1
c=9
```

[ ]:
```
def my_func(*args, **kwargs):
    # your code here

my_func(a,b,c,d=12,e=33)
```

### 15.1 Evaluate Functions-1

[5]:
```
def my_func(*args, **kwargs):

    print("# of positional parameters: {0}\n# of keyword parameters: {1}".format(len(args), len(kwargs)))

my_func(a,b,c,d=12,e=33)
```

```
# of positional parameters: 3
# of keyword parameters: 2
```

## 16 Functions-2

Define a function `sum_of_integers` that can take any number of integer parameters and return the sum of their values

`def sum_of_integers`

[129]:
```
# set up variables
num_integers = np.random.randint(3,12)
input_list = np.random.randint(0,100, size=num_integers)
```

[129]:
```
array([14, 92, 56, 22, 75, 68])
```

[ ]:
```
def sum_of_integers(... your code here)
    # your code here
```

[ ]:
```
print(sum_of_integers(*input_list)) # use *input_list to pass list values as individual params
```

### 16.1 Evaluate Functions-2

[132]:
```
def sum_of_integers(*args):
    integer_sum = reduce(lambda x,y:x+y, args)
    return integer_sum

sum_of_integers(*input_list) # use *input_list to pass list values as individual params
```

[132]:
```
327
```

# 17 File IO-1

1. Read the data in `/home/pi/seneca-prg550-2022-spring/labs/aapl_stock.json`
2. Write to csv file that contains only `date` and `close_price` fields (`/home/pi/seneca-prg550-2022-spring/labs/lecture_test_aapl_stock.csv'`) For example:

```
10/06/2021,142.0
10/05/2021,141.11
10/04/2021,139.14
...
10/11/2011,14.2961
10/10/2011,13.8861
10/07/2011,13.2071
```

Hint: use `json.loads()` to parse each line of `aapl_stock.json` into a dictionary

```
json.loads('{"date": "10/06/2021", "close_price": 142.0, "volume": 83221120, "open": 139.47, "high": 142.15, "low": 138.37}\n')
```

`json.loads()` should return a dictionary:

```
    {'date': '10/06/2021',
     'close_price': 142.0,
     'volume': 83221120,
     'open': 139.47,
     'high': 142.15,
     'low': 138.37}
```

[1]:
```
# show the first 3 lines
!head -n 3 /home/pi/seneca-prg550-2022-spring/labs/aapl_stock.json
```

```
{"date": "10/06/2021", "close_price": 142.0, "volume": 83221120, "open": 139.47,
"high": 142.15, "low": 138.37}
{"date": "10/05/2021", "close_price": 141.11, "volume": 80861060, "open":
139.49, "high": 142.24, "low": 139.36}
{"date": "10/04/2021", "close_price": 139.14, "volume": 98322010, "open":
141.76, "high": 142.21, "low": 138.27}
```

[2]:
```
# show the last 2 lines
!tail -n 2 /home/pi/seneca-prg550-2022-spring/labs/aapl_stock.json
```

```
{"date": "10/10/2011", "close_price": 13.8861, "volume": 440563864, "open":
13.5389, "high": 13.8861, "low": 13.5075}
```

```
{"date": "10/07/2011", "close_price": 13.2071, "volume": 535396186, "open":
13.4207, "high": 13.4907, "low": 13.1603}
```

[3]:
```
# setting variables
input_file_name = '/home/pi/seneca-prg550-2022-spring/labs/aapl_stock.json'
output_file_name = '/home/pi/seneca-prg550-2022-spring/labs/lecture_test_aapl_stock.csv'
```

[ ]:
```
def stock_filter(input_file_name, output_file_name):

    # your code here
    #
```

[ ]:
```
!head -n 3 /home/pi/seneca-prg550-2022-spring/labs/output.csv
```

[ ]:
```
!tail -n 3 /home/pi/seneca-prg550-2022-spring/labs/output.csv
```

## 17.1 Evaluate File-IO-1

[4]:
```
def stock_filter(input_file_name, output_file_name):
    # writing manually
    with open(input_file_name, 'r') as f:
        aapl_json = f.readlines()
    aapl_parsed = [ json.loads(line) for line in aapl_json]

    with open(output_file_name, 'w') as outfile:
        for row in aapl_parsed:
            out_str = f"{row['date']},{row['close_price']}"
            outfile.write(out_str)
            outfile.write('\n')
    return aapl_parsed

def stock_filter2(input_file_name, output_file_name):
    # pandas to write csv
    with open(input_file_name, 'r') as f:
        aapl_json = f.readlines()
    aapl_parsed = [ json.loads(line) for line in aapl_json]
```

```
    df = pd.DataFrame(aapl_parsed)
    df.to_csv(output_file_name, columns=['date','close_price'], index=False, header=False)
    return df
```

[11]:
```
output = stock_filter(input_file_name, output_file_name)
```

[12]:
```
!head -n 3 {output_file_name}
```

```
10/06/2021,142.0
10/05/2021,141.11
10/04/2021,139.14
```

[13]:
```
!tail -n 3 {output_file_name}
```

```
10/11/2011,14.2961
10/10/2011,13.8861
10/07/2011,13.2071
```

[15]:
```
output = stock_filter2(input_file_name, output_file_name)
```

[16]:
```
!head -n 3 /home/pi/seneca-prg550-2022-spring/labs/output.csv
```

```
10/06/2021,142.0
10/05/2021,141.11
10/04/2021,139.14
```

[17]:
```
!tail -n 3 /home/pi/seneca-prg550-2022-spring/labs/output.csv
```

```
10/11/2011,14.2961
10/10/2011,13.8861
10/07/2011,13.2071
```

## 18 Pandas-1

Use the yfinance library to

1. load daily data for stock ticker 'AAPL' between 01-Jan-2021 and 31-Dec-2021
2. plot daily close price
3. plot daily volume

4. calculate the monthly average close price (take each month of close prices and perform average) Hint: you will need to create an additional column and then group-by

Reference: https://github.com/ranaroussi/yfinance

[148]:
```
# set up variables

aapl = yf.Ticker("AAPL")
start_date = '2021-01-01'
stop_date = '2021-12-31'
```

[ ]:
```
# your code here to load stock data
historical_data = ...?
```

[ ]:
```
# your code here to plot daily close price
```

[ ]:
```
# your code here to plot daily volume
```

[ ]:
```
# your code here to calculate monthly average close price
```

## 19 Evaluate Pandas-1

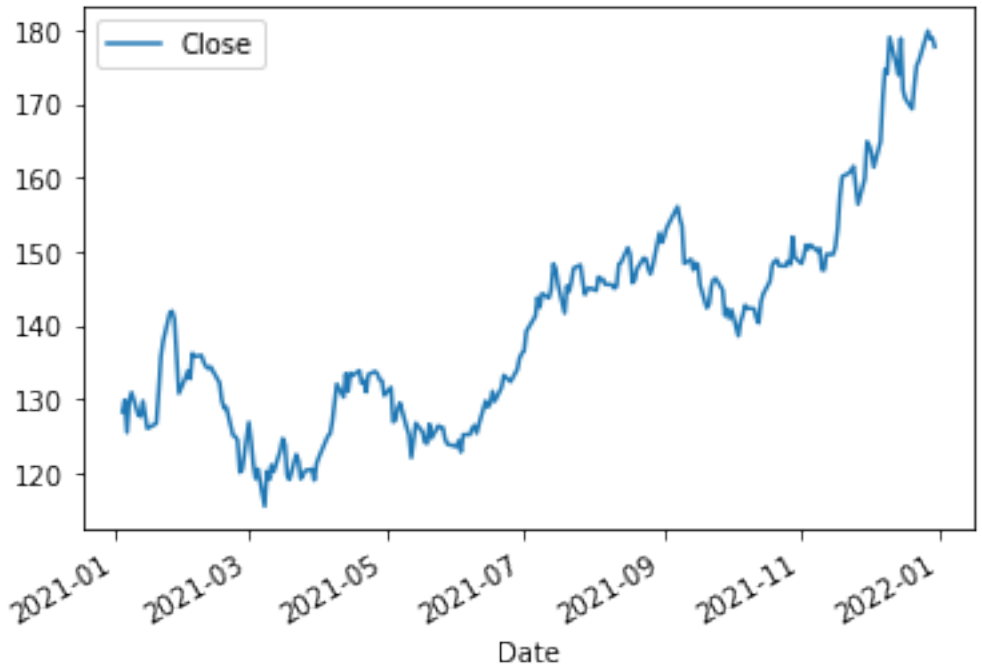
[149]:
```
historical_data = aapl.history(start=start_date, end=stop_date)
historical_data.head(3)
```

[149]:

| Date | Open | High | Low | Close | Volume | Dividends | Stock Splits |
|---|---|---|---|---|---|---|---|
| 2021-01-04 | 132.338619 | 132.427820 | 125.638430 | 128.264984 | 143301900 | 0.0 | 0 |
| 2021-01-05 | 127.749607 | 130.574397 | 127.293671 | 129.850845 | 97664900 | 0.0 | 0 |
| 2021-01-06 | 126.589936 | 129.890474 | 125.261788 | 125.479843 | 155088000 | 0.0 | 0 |

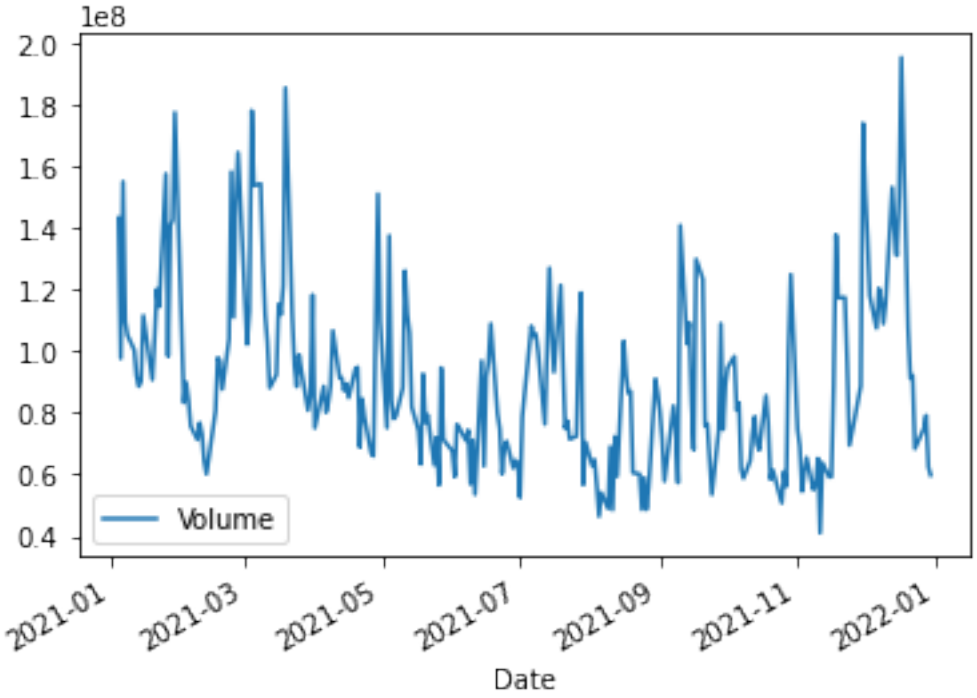
[150]:
```
historical_data.plot(y='Close', use_index=True)
```

[150]: <AxesSubplot:xlabel='Date'>




[151]:
```
historical_data.plot(y='Volume', use_index=True)
```

[151]: <AxesSubplot:xlabel='Date'>




[152]:
```
historical_data['month'] = historical_data.index.month # create month for group-by
historical_data.groupby('month')['Close'].mean()
```

[152]:
```
month
1      131.859744
2      130.520474
3      120.973499
4      130.841815
5      126.020081
6      129.220369
7      144.315016
8      147.516969
```

```
9     147.684634
10    144.953748
11    153.791792
12    172.886818
Name: Close, dtype: float64
```

[ ]:

## 20 Pandas-2

Using the Titanic dataset, answer the questions below

1. How many people surrvived from each Pclass?
2. What was the average ticket price for each Pclass?
3. What is the age of the oldest male and female in each Pclass?
4. What is the age of the yongest male and female in each Pclass?
5. What is the last name of the person with ticket number 17464? Did they survive?
6. Create a scatter plot of Fare paid vs Pclass (ie x=Pclass, y=Fare)

[153]:
```
# run curl at command line to download local version of titanic.csv
!curl https://raw.githubusercontent.com/pandas-dev/pandas/main/doc/data/titanic.csv --output /home/pi/workspace/titanic.
 ↪csv
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 60302  100 60302    0     0   175k      0 --:--:-- --:--:-- --:--:--  176k
```

[30]:
```
# set up variables and import csv into dataframe
input_file = '/home/pi/workspace/titanic.csv'
df = pd.read_csv(input_file)
df.head(3)
```

[30]:
```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3

                                                Name     Sex   Age  SibSp  \
```

```
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
```

[ ]:
```
# How many people survived from each Pclass?

# your code here
```

[ ]:
```
# What was the average ticket price for each Pclass?

# your code here
```

[ ]:
```
# What is the age of the oldest male and female in each Pclass?
# your code here
```

[ ]:
```
# What is the age of the yongest male and female in each Pclass?
# your code here
```

[ ]:
```
# What is the last name of the person with ticket number 17464? Did they survive?
# your code here
```

[ ]:
```

```

## 20.1 Evaluate Pandas-2

[181]:
```
df.groupby('Pclass')['Survived'].sum()
```

[181]:
```
Pclass
1    136
2     87
3    119
Name: Survived, dtype: int64
```

[182]:
```
df.groupby('Pclass')[['Survived', 'Fare']].agg({'Survived':'sum', 'Fare':'mean'})
```

[182]:

| Pclass | Survived | Fare |
|---|---|---|
| 1 | 136 | 84.154687 |
| 2 | 87 | 20.662183 |
| 3 | 119 | 13.675550 |

[183]:
```
df.groupby(['Pclass','Sex'])[['Age']].agg({'Age':['min','max']})
```

[183]:

| | | Age | |
|---|---|---|---|
| | | min | max |
| Pclass | Sex | | |
| 1 | female | 2.00 | 63.0 |
| | male | 0.92 | 80.0 |
| 2 | female | 2.00 | 57.0 |
| | male | 0.67 | 70.0 |
| 3 | female | 0.75 | 63.0 |
| | male | 0.42 | 74.0 |

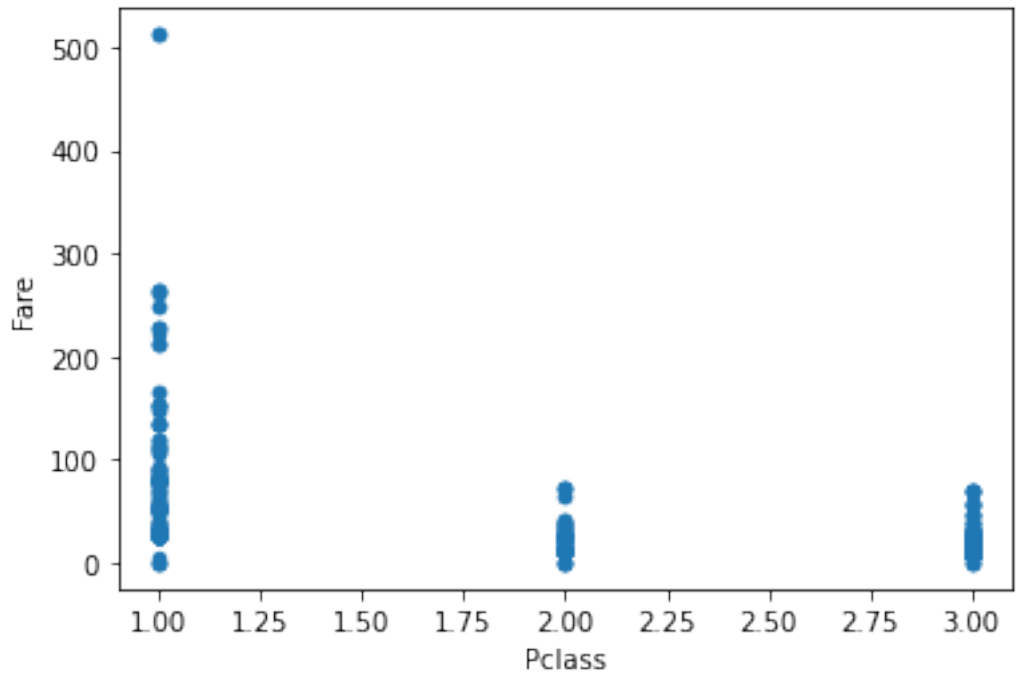
[26]:
```
# What is the last name of the person with ticket number 17464? Did they survive?
df.loc[df['Ticket']=='17464']
```

[26]:

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 457 | 458 | 1 | 1 | Kenyon, Mrs. Frederick R (Marion) | female | NaN | 1 | 0 | 17464 | 51.8625 | D21 | S |

[185]:
```
df.plot(x='Pclass', y='Fare', kind='scatter')
```

[185]:
```
<AxesSubplot:xlabel='Pclass', ylabel='Fare'>
```




[ ]:

## 21 BMP280-1

1. Connect the BMP280 to your Pi
2. Create a csv file (`/home/pi/workspace/BMP280_Lecture_Test_a.csv`) with 100 lines of captured data from the BMP280 `time,temperature,pressure 2022-07-19 01:27:07,26.5021484375,987.5028677231073 2022-07-19 01:27:07,26.5095703125,987.4675822492511 2022-07-19 01:27:08,26.516796875,987.4492497902675 2022-07-19 01:27:09,26.5267578125,987.4760624275071`
3. To visualize how the measurements change over time, plot a time-series of the temperature and pressure (ie plot with x-axis=time, y-axis=temperature)
4. To visualize the correlation between temperature and pressure, create a plot with x=temperature, y=pressure

[169]:
```
# set up variables
import time
import board
import adafruit_bmp280
from datetime import datetime


i2c = board.I2C()
bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c) # use default address
bmp280.sea_level_pressure = 1000.85  # set on 22/07/05 @ 15h20, sea level pressure in Toronto, in hPa

bmp_output_file = '/home/pi/workspace/BMP280_Lecture_Test_a.csv'
header='time,temperature,pressure'
```

[ ]:
```
# your code here to create csv file
```

[ ]:
```
# your code here to load csv into dataframe
```

[ ]:
```
# your code here to create temperature vs time plot
```

[ ]:
```
# your code here to create pressure vs time plot
```

### 21.1 Evaluate BMP280-1

[171]:
```
# setup
import time
import board
import adafruit_bmp280
from datetime import datetime


i2c = board.I2C()
bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c) # use default address
bmp280.sea_level_pressure = 1000.85  # set on 22/07/05 @ 15h20, sea level pressure in Toronto, in hPa

bmp_output_file = '/home/pi/workspace/BMP280_Lecture_Test_a.csv'
header='time,temperature,pressure'
with open(bmp_output_file, 'w') as f:
```

```
    f.write(header+'\n')
    for idx in range(0,100):
        time_str = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        data_str = f"{time_str},{bmp280.temperature},{bmp280.pressure}"
        f.write(data_str+'\n')
        time.sleep(0.5)
```

[172]:
```
df = pd.read_csv(bmp_output_file)
```

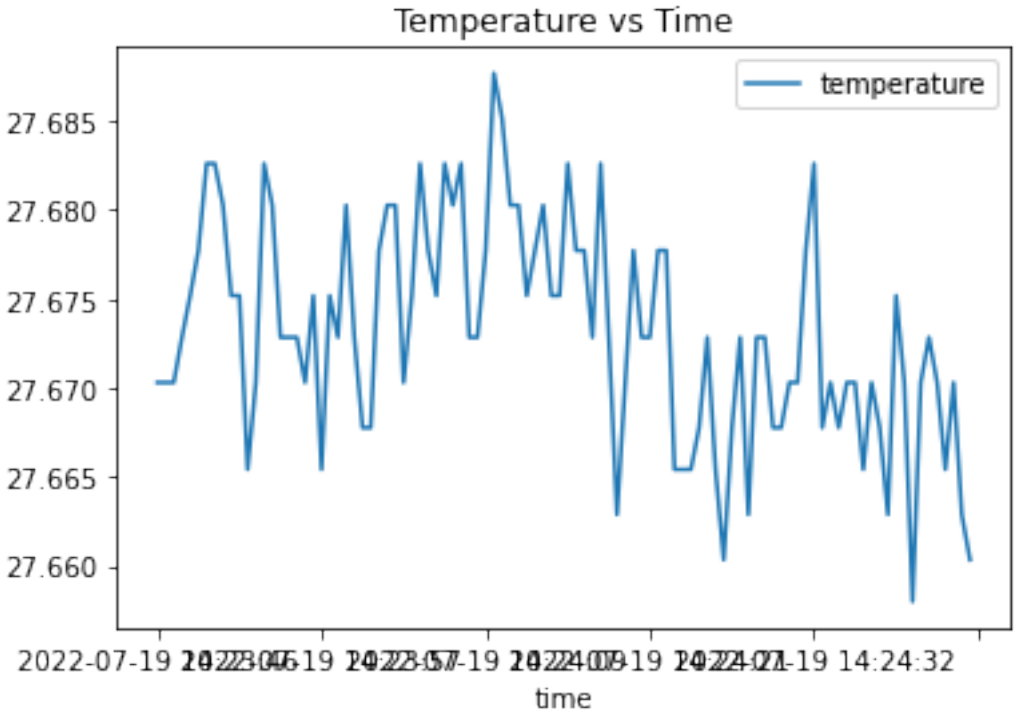
[173]:
```
!head -n 5 /home/pi/workspace/BMP280_Lecture_Test_a.csv
```

```
time,temperature,pressure
2022-07-19 14:23:46,27.6703125,987.0600693898718
2022-07-19 14:23:46,27.6703125,987.0309293008388
2022-07-19 14:23:47,27.6703125,987.034837496552
2022-07-19 14:23:48,27.6728515625,987.0853676758961
```

[174]:
```
df.set_index('time', inplace=True)
```

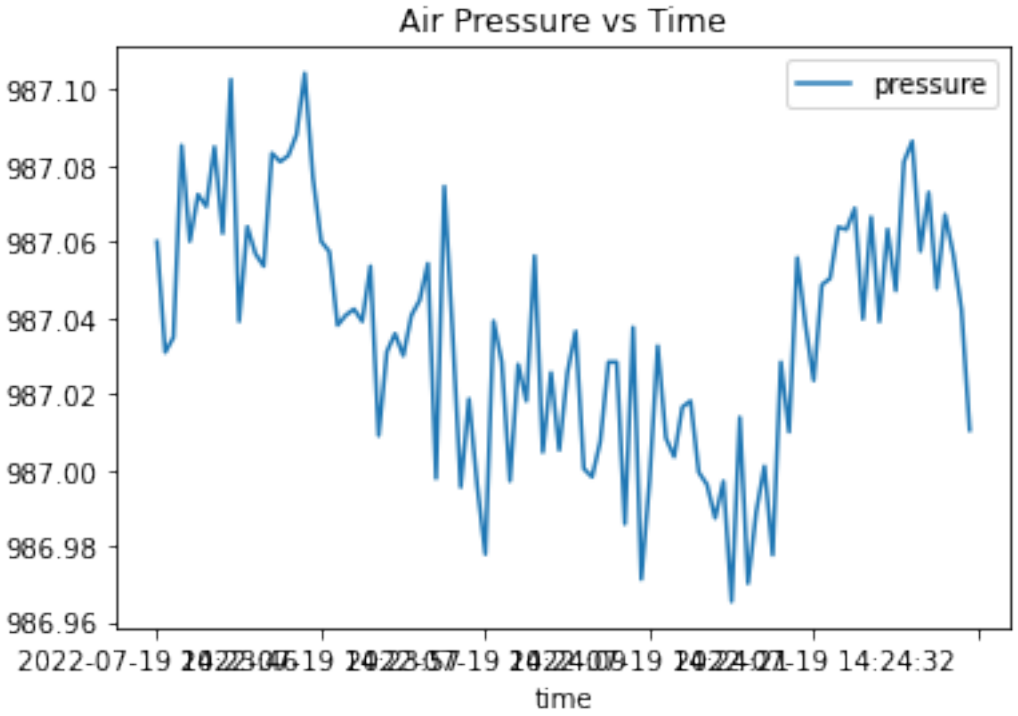
[175]:
```
df.plot(y='temperature', use_index=True, title='Temperature vs Time')
```

[175]:
```
<AxesSubplot:title={'center':'Temperature vs Time'}, xlabel='time'>
```



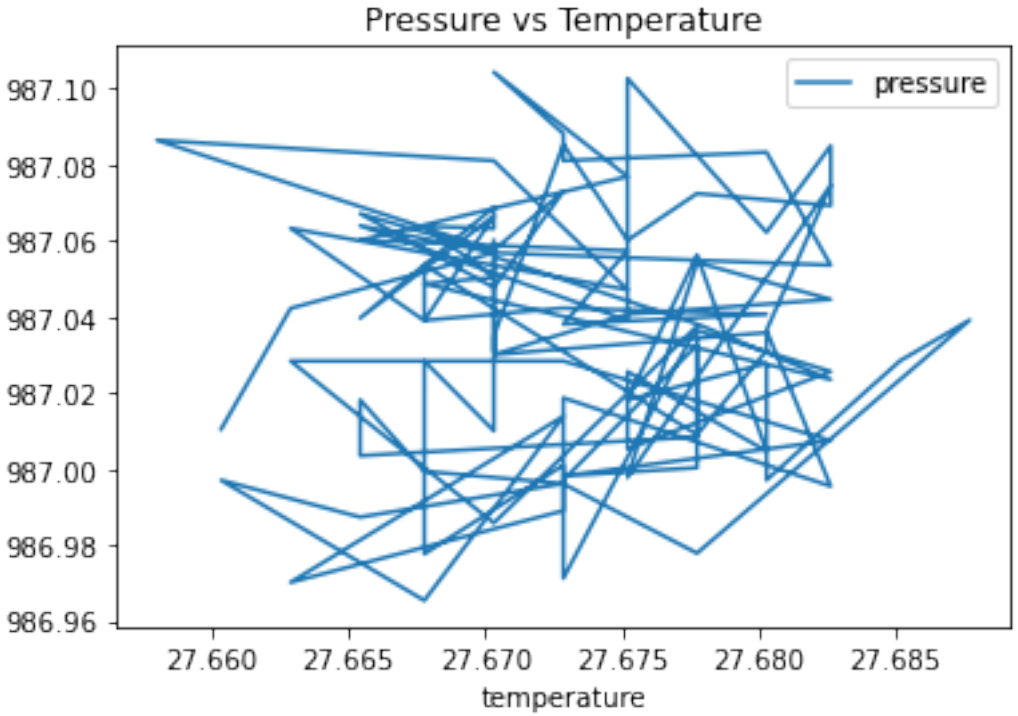

[176]:
```
df.plot(y='pressure', use_index=True, title='Air Pressure vs Time')
```

[176]: <AxesSubplot:title={'center':'Air Pressure vs Time'}, xlabel='time'>




[177]:
```
df.plot(x='temperature', y='pressure', title='Pressure vs Temperature')
```

[177]: <AxesSubplot:title={'center':'Pressure vs Temperature'}, xlabel='temperature'>




## 22 BMP280-2

1. Connect the BMP280 to your Pi
2. Create a csv file (`/home/pi/workspace/BMP280_Lecture_Test_b.csv`) with 100 lines of captured data from the BMP280 `time,pressure,temperature 2022-07-19 01:34:59,987.5613893827572,26.6798828125 2022-07-19 01:35:00,987.4763473135786,26.6921875 2022-07-19 01:35:01,987.5157352877336,26.6921875 2022-07-19 01:35:01,987.4833945602606,26.682421875`
3. To visualize how the measurements change over time, plot a time-series of the temperature and pressure (ie plot with x-axis=time, y-axis=temperature)
4. To visualize the correlation between temperature and pressure, create a plot with x=pressure, y=temperature

[ ]:
```
# set up variables
import time
import board
import adafruit_bmp280
from datetime import datetime


i2c = board.I2C()
bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c) # use default address
bmp280.sea_level_pressure = 1000.85  # set on 22/07/05 @ 15h20, sea level pressure in Toronto, in hPa

bmp_output_file = '/home/pi/workspace/BMP280_Lecture_Test_b.csv'
header='time,pressure,temperature'
```

[ ]:
```
# your code here to create csv file
```

[ ]:
```
# your code here to load csv into dataframe
```

[ ]:
```
# your code here to create temperature vs time plot
```

[ ]:
```
# your code here to create pressure vs time plot
```

## 23 Evaluate BMP280-2

[ ]:
```
import time
import board
import adafruit_bmp280
from datetime import datetime


i2c = board.I2C()
bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c) # use default address
bmp280.sea_level_pressure = 1000.85  # set on 22/07/05 @ 15h20, sea level pressure in Toronto, in hPa

bmp_output_file = '/home/pi/workspace/BMP280_Lecture_Test_b.csv'
header='time,pressure,temperature'
with open(bmp_output_file, 'w') as f:
```

```
    f.write(header+'\n')
    for idx in range(0,100):
        time_str = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        data_str = f"{time_str},{bmp280.pressure},{bmp280.temperature}"
        f.write(data_str+'\n')
        time.sleep(0.5)
```

[ ]:
```
!head -n 5 /home/pi/workspace/BMP280_Lecture_Test_b.csv
```

[ ]:
```
df.set_index('time', inplace=True)
```

[ ]:
```
df.plot(y='temperature', use_index=True, title='Temperature vs Time')
```

[ ]:
```
df.plot(y='pressure', use_index=True, title='Air Pressure vs Time')
```

[ ]:
```
df.plot(x='pressure', y='temperature', title='Temperature vs Pressure')
```

[ ]:
```

```