Python Coding Cheat-Sheet

Made by Alex Rudyak
Code Blocks theme - tomorrow-night-blue

General

Make a copy of a list (unlinked) (shallow copy)

```
listA = [0, 1, 2, 3]
listB = listA[:]
```

Reverse an Integer

4562 → 2654

```
n = 4562
rev = 0

while(n > 0):
    a = n % 10
    rev = rev * 10 + a
    n = n // 10

print(rev)
```

Reverse a String

```
text = "hello, world"
reversed_text = text[::-1]
```

Combine two lists into a dictionary

```
test_keys = []
test_values = []
dic = {}
for key in test_keys:
    for value in test_values:
        dic[key] = value
        test_values.remove(value)
        break
print (dic)
```

Check whether a char is int

```
string = "avs12ssd"
for char in string:
    if char.isnumeric():
       # char is an integer
```

Output list elements as string with spaces

[1, hi, 2, four] → “1 hi 2 four”

```
list = [1, hi, 2, four]
string = " ".join(list)
print (string) # "1 hi 2 four"
```

Convert int to hexadecimal including negatives

```
num = 10
hexaNum = format((num + (1<<32)) % (1<<32),'x') # 1<<32 is a bit wise operation to
move the bit "1" 32 steps to The left.
print (hexaNum) # A
```

Find longest common prefix in a list of strings

[“flower”,”flow”,”flight] → “fl”

```
strs = ["flower","flow","flight"]
if not strs: return ""
        prefix = strs[0]
        for i in range(len(strs)):
            while strs[i].find(prefix): # find the first index of prefix
                prefix = prefix[0:len(prefix)-1]
                if not prefix:
                    return ""
        return prefix
```

Iterate through index and value using enumerate

```
doc_list = "Adventures of Arch","Bamby","Princess and the pea"
# Iterate through the indices (i) and elements (doc) of documents
```

```
    for i, doc in enumerate(doc_list):
           #  i = 0, doc = "Adventures of Arch"
           #  i = 1, doc = "Bamby"
           #  i = 2, doc = "Princess and the pea"
```

Check whether input string is a valid IP

```
def is_valid_IP(strng):
    ip_cells = strng.split(".") # split the string to 4 cells like ipv4
    if len(ip_cells) == 4: # if the length is not 4 its not a valid ip
        for cell in ip_cells:
            if cell.isnumeric(): # if the cell is not numeric its not a valid ip
                if 0 <= int(cell) <= 255: # the values should be between 0 and 255
                    state = True
                else:
                    return False
            else:
                return False
            if len(cell) == 3: # if its a 3 digit number and starts with 0 its not
valid (although I think it is)
                if cell.startswith("0"):
                    return False
        return state
    else:
        return False
```

Find Square Root - Bi-section Search

```
x = 25
epsilon = 0.001
numGuess = 0
low = 1.0
high = x
guess = (high + low)/2.0

while abs(guess**2 - x) >= epsilon:
    print('low = ' + str(low) + ' high = ' + str(high) + ' | Our guess: ' +
str(guess))
    numGuess += 1
    if guess**2 < x:
        low = guess
    else:
        high = guess
    guess = (high + low)/2.0

print('Number of Guesses = ' + str(numGuess))
print(str(guess) + ' is Close to square root of ' + str(x))
```

Guess the number game - Bi-section search

```
print("Please think of a number between 0 and 100!")
high = 100
low = 0
first_guess = (high - low)//2
user_input = ""
print("Is your secret number " + str(first_guess) + " ?")

while user_input != 'c':

    user_input = input("Enter 'h' to indicate the guess is too high. Enter 'l' to
indicate the guess is too low. Enter 'c' to indicate I guessed correctly. ")

    if user_input == 'h':
        high = first_guess
    elif user_input == 'l':
        low = first_guess
    elif user_input == 'c':
        print("Game over. Your secret number was: " + str(first_guess))
    else:
        print("Sorry, I did not understand your input.")

    first_guess = (high + low)//2

    if user_input != 'c':
        print("Is your secret number " + str(first_guess) + " ?")
```

Greatest Common Divisor - Loop

```
def gcdIter(a, b):
    '''
    a, b: positive integers

    returns: a positive integer, the greatest common divisor of a & b.
    '''
    tmp = a if a<b else b
    while tmp > 0:
        if ((a%tmp == 0) and (b%tmp == 0)):
            return tmp
        tmp -= 1
```

Greatest Common Divisor - Recursion

```
def gcdRecur(a, b):
    '''
    a, b: positive integers

    returns: a positive integer, the greatest common divisor of a & b.
    '''
    if b == 0:
        return a
    else:
        return gcdRecur(b, a%b)
```

Check whether a String is a Palindrome - Recursion

```
def isPal(s):
    if len(s) < 1:
        return True
    else:
        return s[0] == s[-1] and isPal(s[1:-1])
```

Print every other element in a tuple to another tuple

```
def oddTuples(aTup):
    '''
    aTup: a tuple

    returns: tuple, every other element of aTup.
    '''
    new_t = ()
    for num,t in enumerate(aTup):
        if num%2 == 0:
            new_t += (t,)
    return new_t
```

```
print(oddTuples(('I', 'am', 'a', 'test', 'tuple')))
```

Special class methods

```
class Coordinate (object):

    def __init__(self, x, y): # Defines the initialization of the class object
        self.x = x
        self.y = y

    def __str__(self): # Defines print of object | print(self)
        return "<"+str(self.x)+","+str(self.y)+">"

    def __add__(self,other): # Defines addition | self + other
        return Coordinate(self.x + other.x, self.y + other.y)

    def __sub__(self,other): # Defines subtraction | self - other
        return Coordinate(self.x - other.x, self.y - other.y)
```

Matplotlib

Pandas - Data manipulation

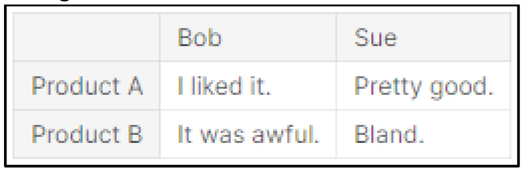
Importing the library

```
import pandas as pd
```

Create a DataFrame (Table)

```
pd.DataFrame({'Bob': ['I liked it.', 'It was awful.'], # column 1
              'Sue': ['Pretty good.', 'Bland.']}, # column 2
             index = ['Product A', 'Product B']) # index (row names)
```

We get:



	Bob	Sue
Product A	I liked it.	Pretty good.
Product B	It was awful.	Bland.

Create a Series (Single column of a Table)

```
pd.Series([30, 35, 40], index=['2015 Sales', '2016 Sales', '2017 Sales'],
name='Product A')
```

We get:

```
2015 Sales    30
2016 Sales    35
2017 Sales    40
Name: Product A, dtype: int64
```

Import data from a csv

Given a location with a csv file, import it into our workspace

```
# Path of the file to read
fifa_filepath = "../input/fifa.csv"

# Read the file into a variable fifa_data
fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dates=True)
```

Change the datatype of a column

Changes the data type of column “*points*” into a *float64* type.

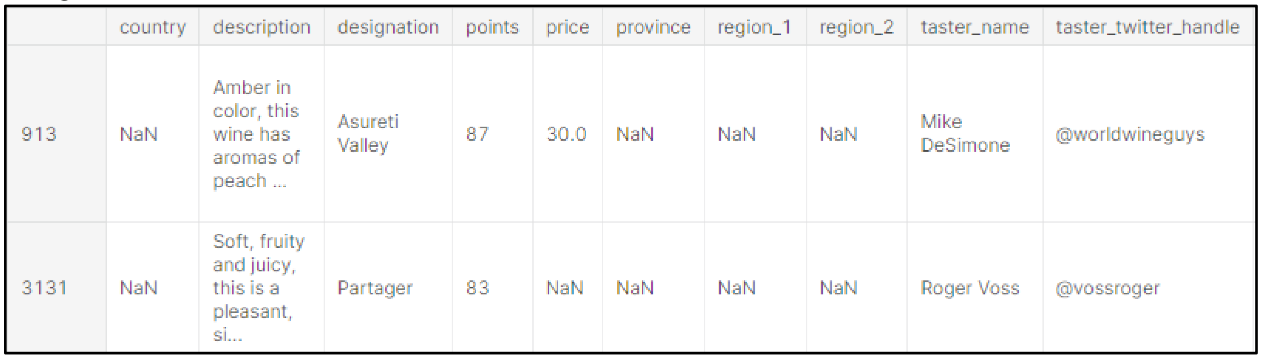
```
reviews.points.astype('float64')
```

Check whether we have missing data in our table

Given table “*Reviews*” get all the rows of the NaN values in “*Country*”.

```
reviews[pd.isnull(reviews.country)]
```

We get:



	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle
913	NaN	Amber in color, this wine has aromas of peach ...	Asureti Valley	87	30.0	NaN	NaN	NaN	Mike DeSimone	@worldwineguys
3131	NaN	Soft, fruity and juicy, this is a pleasant, si...	Partager	83	NaN	NaN	NaN	NaN	Roger Voss	@vossroger

Change column name in a table

Given a table “*Reviews*” with a column named “*points*”, change its name to “*score*”.

```
reviews.rename(columns={'points': 'score'})
```

Change Index names (row and column)

Given table “*Reviews*” change its rows index name to “*wines*” and column index name to “*fields*”.

```
reviews.rename_axis("wines", axis='rows').rename_axis("fields", axis='columns')
```

Concatenate two tables together

Given two tables “*canadian_youtube*” and “*british_youtube*” concatenate them side by side, as in “stick” them together basically adding more rows while keeping the same columns.

```
canadian_youtube = pd.read_csv("../input/youtube-new/CAvideos.csv")
british_youtube = pd.read_csv("../input/youtube-new/GBvideos.csv")
pd.concat([canadian_youtube, british_youtube])
```

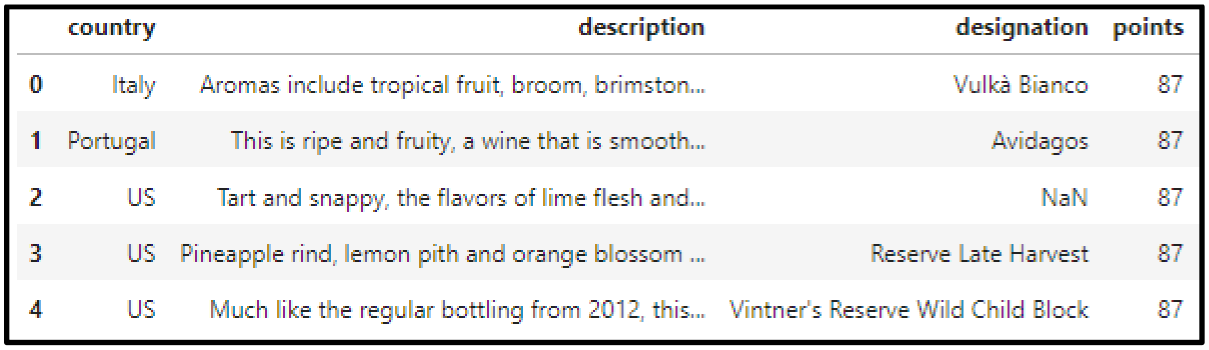
Merge two table together that has the same column

Given two tables “*Powerlifting_meets*” and “*Powerlifting_competitors*” that have the same column “*MeetID*” we can merge that together based on the MeetID index.
To make sure every column in the merged table has unique names we add suffixes to each column of the original tables, “*_meets*” for the first table and “*_competitors*” for the second.

```
left = powerlifting_meets.set_index(["MeetID"])
right = powerlifting_competitors.set_index(["MeetID"])
powerlifting_combined = left.join(right, lsuffix='_meets',
rsuffix='_competitors')
```

Count how many times have a word came up in a table column

Given a table “*Reviews*”, find how many times have the words “*fruity*” and “*tropical*” appear in the description column.



	country	description	designation	points
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87
2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87
3	US	Pineapple rind, lemon pith and orange blossom ...	Reserve Late Harvest	87
4	US	Much like the regular bottling from 2012, this...	Vintner's Reserve Wild Child Block	87

```
n_trop = reviews.description.map(lambda desc: "tropical" in desc).sum() # tropical
word frequency

n_fruity = reviews.description.map(lambda desc: "fruity" in desc).sum() # fruity
word frequency

descriptor_counts = pd.Series([n_trop, n_fruity], index=['tropical', 'fruity'])
```

Apply a function on every column in a table

Given a table “*Reviews*” with a variable “*country*” and “*points*” in it, change every column based on the criteria in the function.

```
def stars(row):
    if row.country == 'Canada':
        return 3
    elif row.points >= 95:
        return 3
    elif row.points >= 85:
        return 2
    else:
        return 1

star_ratings = reviews.apply(stars, axis='columns')
```

Find values from a table based on grouping in the table

Given table “*Reviews*”, we slice the table by “points” and take the minimum price.

```
reviews.groupby('points').price.min()
```

Sort table by particular parameter

Given table “*Countries_reviewed*”, sort the table by parameter “*len*”.

This will keep the original Indexes!

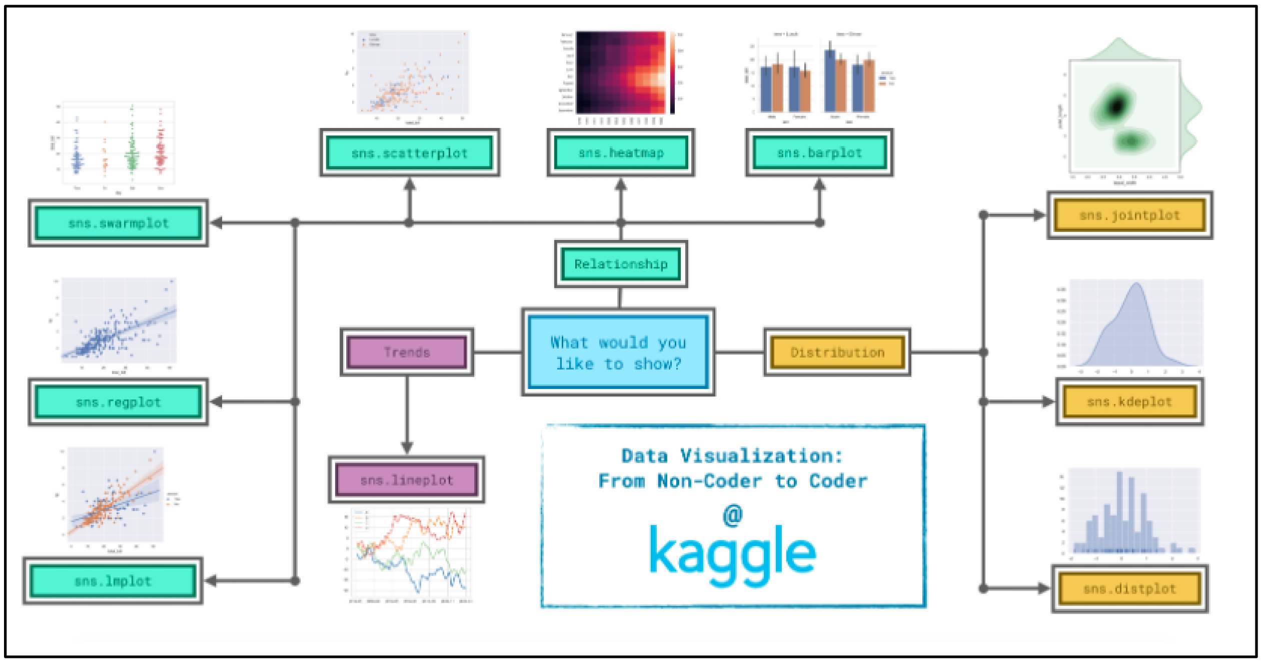
Index | Name   Index | Name
—---------------   —---------------
1 | Bob   sort_values(by = ‘name’)   2 | Alex
—---------------   →   —---------------
2 | Alex   1 | Bob
—---------------   —---------------

```
countries_reviewed.sort_values(by = 'len')
```

The default order is ascension, starting from the smallest number at [0] and largest number at [end]. To change that we can add the “ascending = False” key.

```
countries_reviewed.sort_values(by = 'len', ascending = False)
```

Seaborn - Data needs visualizing as well



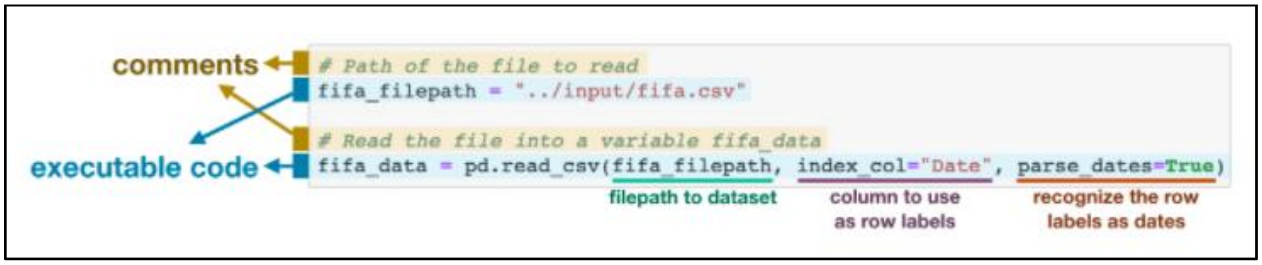
Importing the library

```
import seaborn as sns
```

To import the actual data we will use **[Pandas](#)**, the library discussed here as well.

```
# Path of the file to read
fifa_filepath = "../input/fifa.csv"

# Read the file into a variable fifa_data
fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dates=True)
```

Explained as:



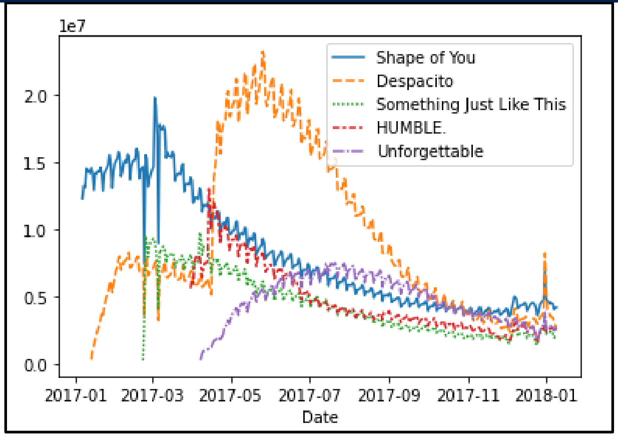
```
# Path of the file to read
fifa_filepath = "../input/fifa.csv"

# Read the file into a variable fifa_data
fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dates=True)
```

comments
executable code
filepath to dataset
column to use as row labels
recognize the row labels as dates

Plot a line chart

Given CSV data "*spotify_data*" we can plot it using the following command, it will automatically use the table's left column as **X axis** and **Y axis** as the **values of the cell they are in.**
The table's upper index is used as a **Legend**.

```
# Line plot
sns.lineplot(data=spotify_data)
```



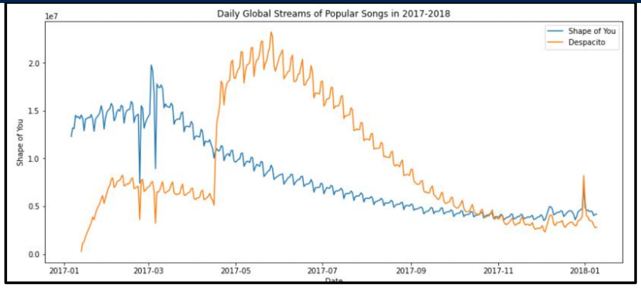
Changing the parameters of the plots

```
# Set the width and height of the figure
plt.figure(figsize=(14,6))
# Add title
plt.title("Daily Global Streams of Popular Songs in 2017-2018")
# Add label for horizontal axis
plt.xlabel("Date")
# Add label for vertical axis
```

```
plt.ylabel("Arrival delay (in minutes)")
# Add a Legend to the data
plt.legend()
# Change the style of the figure to the "dark" theme
sns.set_style("dark") # darkgrid || whitegrid || dark || white || ticks
```

Draw a line plot of only one column of data from a table

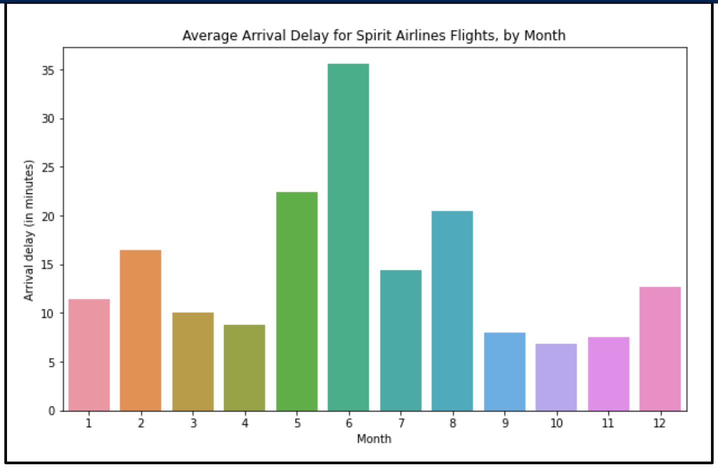
```
# Line chart showing only one column of the table
sns.lineplot(data=spotify_data['Shape of You'], label="Shape of You")
```



Plot a bar chart

Given table "*flight_data*" use the left index of the table as **X axis** and the **Y axis** plot values that **correspond to those indexes and their height.**

```
# Bar chart showing a single column in a table
sns.barplot(x=flight_data.index, y=flight_data['NK'])
```



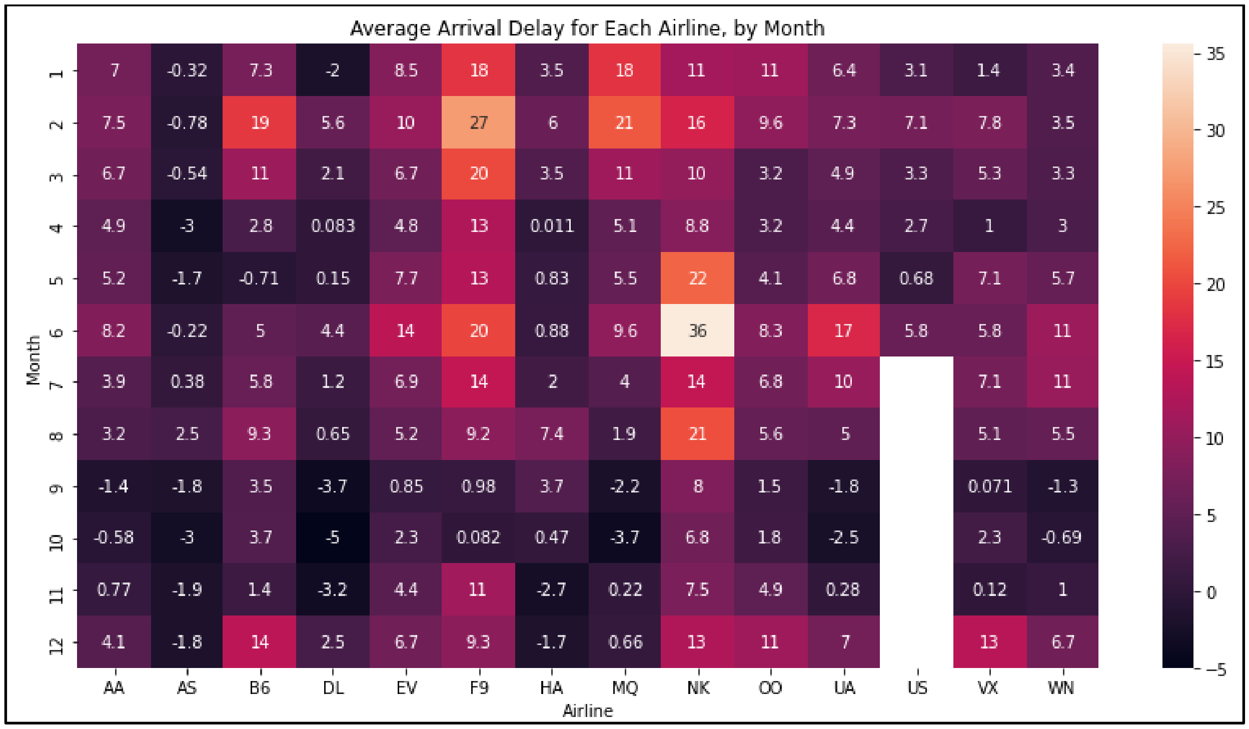
#This_Color_Pallete_is_beautiful.

Plot a heatmap (Spectrogram?!)

Given a table "*flight_data*" plot a heatmap of the data. This also resembles a **Spectrogram.**

Basically turning the whole table into a color book.

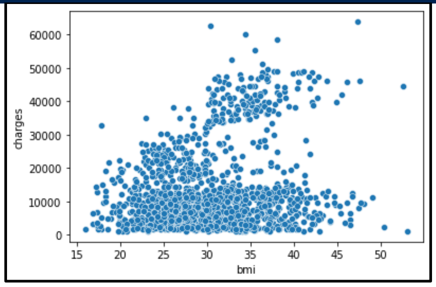
```
# Heatmap showing average arrival delay for each airline by month
sns.heatmap(data=flight_data, annot=True)
```



Plot a scatter plot

Given table "*insurance_data*", plot its "*bmi*" column as **X axis** and "*charges*" column as **Y axis.**

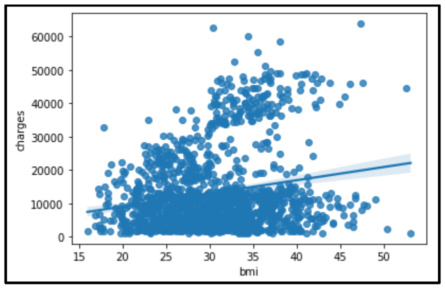
```
sns.scatterplot(x = insurance_data['bmi'], y = insurance_data['charges'])
```



Plot a scatter plot with a regression line

Changing "*scatterplot*" to "regplot" adds a regression line to the scatter plot.

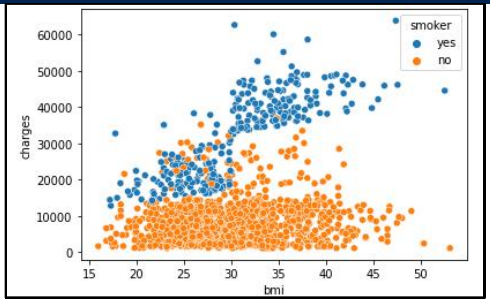
```
sns.regplot(x = insurance_data['bmi'], y = insurance_data['charges'])
```



Color the data set

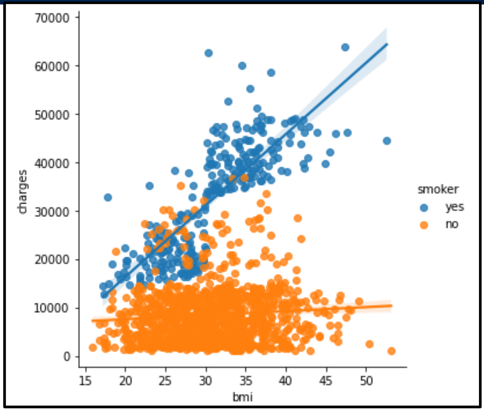
You can color the data set based on column values, to do so we add the "*hue*" keyword. Because the "*smoker*" column has only 2 values, we get only 2 colors.

```
sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'],
hue=insurance_data['smoker'])
```



Adding regression lines based on sub-data

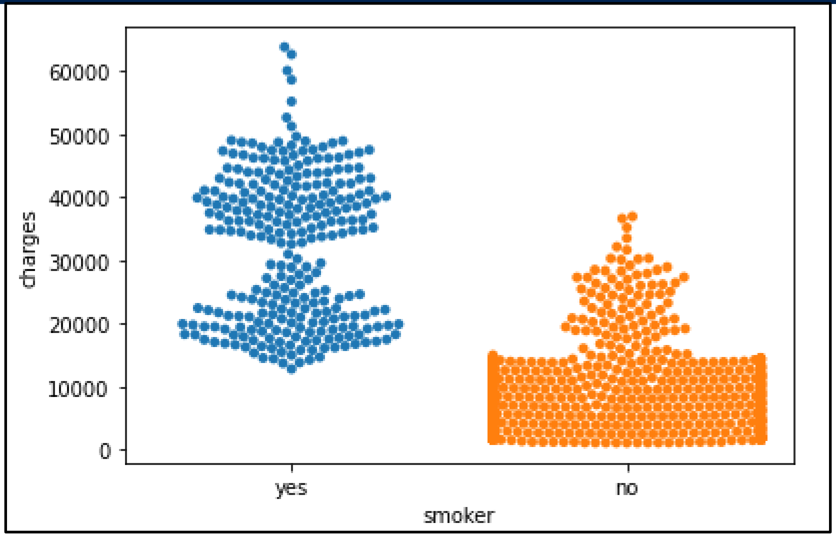
```
sns.lmplot(x="bmi", y="charges", hue="smoker", data=insurance_data)
```



Plot a categorical scatter plot - swarmplot

Instead of **continuous** data as the **X axis** we use **discrete data** ('1' or '0') to plot.
The result is a little bit weird but we can get that the max charges are people who got "yes" ('1') and minimum charges are people that got "no" ('0').

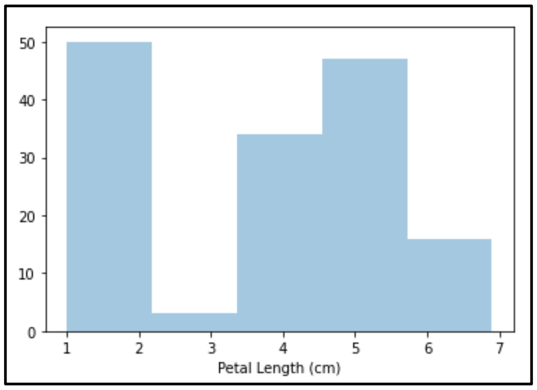
```
sns.swarmplot(x=insurance_data['smoker'], y=insurance_data['charges'])
```



Plot a Histogram

Given table "*iris_data*" plot a histogram with column "*Petal length (cm)*" as **X axis.**

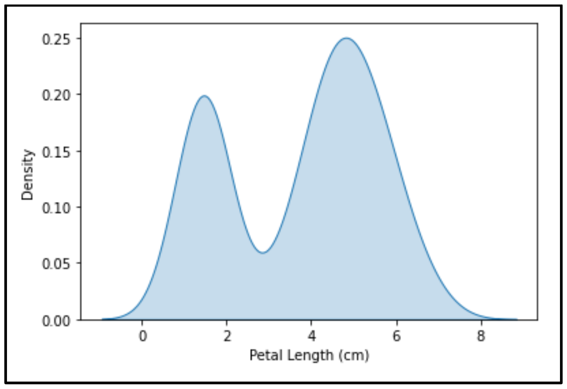
```
# Histogram
sns.distplot(a=iris_data['Petal Length (cm)'], kde=False)
```



Plot a Density plot (KDE)

Given table *"iris_data"*, plot column *"Petal Length (cm)"* as **X axis**.
To color the underneath graph we add *"shade"* keyword.

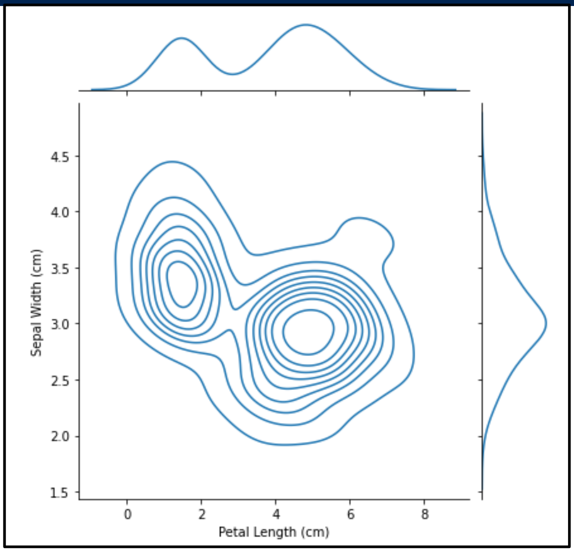
```
# KDE plot
sns.kdeplot(data=iris_data['Petal Length (cm)'], shade=True)
```



Plot a 2D KDE plot

Given table *“iris_data”,* plot column *“Petal Length (cm)”* as a **KDE plot** on the **X axis**, and column *“Sepal Width (cm)”* as a **KDE plot** on the **Y axis**.
Combine (join) them on a single plot.

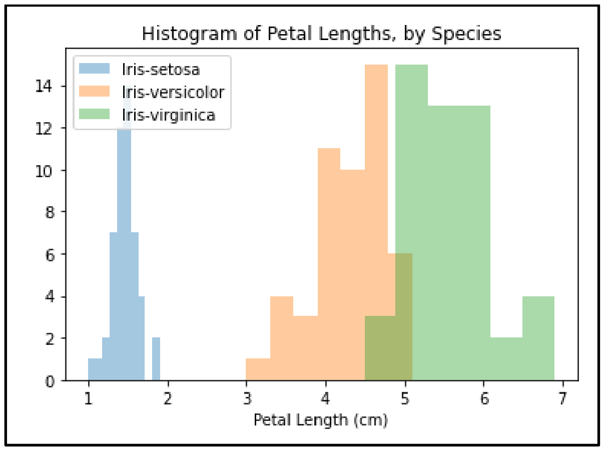
```
sns.jointplot(x=iris_data['Petal Length (cm)'], y=iris_data['Sepal Width (cm)'],
kind="kde")
```



Adding two or more histograms on one plot

Simply calling the histogram function again will add another plot to the existing one.

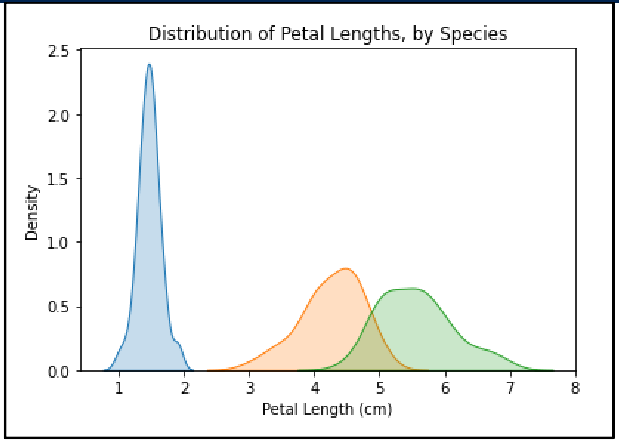
```
sns.distplot(a=iris_set_data['Petal Length (cm)'], label="Iris-setosa", kde=False)
sns.distplot(a=iris_ver_data['Petal Length(cm)'],label="Iris-versicolor",kde=False)
sns.distplot(a=iris_vir_data['Petal Length (cm)'],label="Iris-virginica",kde=False)
```



Adding two or more KDE on one plot

Simply calling the function again adds another plot to the figure.

```
sns.kdeplot(data=iris_set_data['PetalLength(cm)'],label="Iris-setosa",shade=True)
sns.kdeplot(data=iris_ver_data['PetalLength(cm)'],label="Iris-
versicolor",shade=True)
sns.kdeplot(data=iris_vir_data['PetalLength(cm)'],label="Iris-
virginica",shade=True)
```



MITx 6.00.1x

Problem Set 1

Problem 1 - Vowel Count

Assume `s` is a string of lower case characters.

Write a program that counts up the number of vowels contained in the string `s`. Valid vowels are: '`a`', '`e`', '`i`', '`o`', and '`u`'. For example, if `s = 'azcbobobegghakl'`, your program should print:

```
Number of vowels: 5
```

```
vowel_cnt = 0
for char in s: # iterate over chars in sentence
    if char =='a' or char =='e' or char =='i' or char =='o' or char =='u':
        vowel_cnt += 1
print("Number of vowels: ", vowel_cnt)
```

Can make it shorter with:

```
vowl_string = "aeiou"
If char in vowl_string
```

Problem 2 - String in String (with dupe)

Assume `s` is a string of lower case characters.

Write a program that prints the number of times the string `'bob'` occurs in `s`. For example, if `s = 'azcbobobegghakl'`, then your program should print

```
Number of times bob occurs is: 2
```

```python
cnt = 0
for i in range(3, len(s)+1):
    temp = s[i-3:i]
    if temp == 'bob':
        cnt += 1
print("Number of times bob occurs is: ",cnt)
```

Problem 3 - Longest Substring Alphabetically

Assume `s` is a string of lower case characters.

Write a program that prints the longest substring of `s` in which the letters occur in alphabetical order. For example, if `s = 'azcbobobegghakl'`, then your program should print

```
Longest substring in alphabetical order is: beggh
```

In the case of ties, print the first substring. For example, if `s = 'abcbcd'`, then your program should print

```
Longest substring in alphabetical order is: abc
```

```
prefix = s[0]
alphabet = "abcdefghijklmnopqrstuvwxyz"
output = ""

for i in range(len(s)-1):
    if alphabet.find(s[i]) <= alphabet.find(s[i+1]): # find the first index
in a string
# can be replaced with | if s[i] <= s[i+1] because python automatically
measures strings by their ascii but was kept for readability
        prefix += s[i+1]
    elif len(output) < len(prefix):
        output = prefix
        prefix = s[i+1]
    else:
        prefix = s[i+1]

if len(output) < len(prefix):
    output = prefix

# go into "if" only if output is empty
if not output:
    output = prefix

print(output)
```

Problem Set 2

Problem 1

Write a program to calculate the credit card balance after one year if a person only pays the minimum monthly payment required by the credit card company each month.The following variables contain values as described below:

1. `balance` - the outstanding balance on the credit card
2. `annualInterestRate` - annual interest rate as a decimal
3. `monthlyPaymentRate` - minimum monthly payment rate as a decimal

For each month, calculate statements on the monthly payment and remaining balance. At the end of 12 months, print out the remaining balance.

Be sure to print out no more than two decimal digits of accuracy - so print

```
Remaining balance: 813.41
```

instead of

```
Remaining balance: 813.4141998135
```

So your program only prints out one thing: the remaining balance at the end of the year in the format:

```
Remaining balance: 4784.0
```

A summary of the required math is found below:

Monthly interest rate= (Annual interest rate) / 12.0

Minimum monthly payment = (Minimum monthly payment rate) x (Previous balance)

Monthly unpaid balance = (Previous balance) - (Minimum monthly payment)

Updated balance each month = (Monthly unpaid balance) + (Monthly interest rate x Monthly unpaid balance)

```
def endOfYearBalance(balance,annualInterestRate,monthlyPaymentRate):
    updatedBalanceEachMonth = balance
    monthlyInterestRate = annualInterestRate/12.0
    for i in range (12):
        minimumMonthlyPayment = monthlyPaymentRate*updatedBalanceEachMonth
        monthlyUnpaidBalance = updatedBalanceEachMonth - minimumMonthlyPayment
        updatedBalanceEachMonth = monthlyUnpaidBalance +
(monthlyInterestRate*monthlyUnpaidBalance)
    print("Remaining balance: {}".format(round(updatedBalanceEachMonth,2)))
```

```
'''
balance = 42
annualInterestRate = 0.2
monthlyPaymentRate = 0.04
'''
endOfYearBalance(42,0.2,0.04)
'''
balance = 484
annualInterestRate = 0.2
monthlyPaymentRate = 0.04
'''
endOfYearBalance(484,0.2,0.04)
```

Problem 2

Now write a program that calculates the minimum **fixed** monthly payment needed in order to pay off a credit card balance within 12 months. By a fixed monthly payment, we mean a single number which does not change each month, but instead is a constant amount that will be paid each month.

In this problem, we will *not* be dealing with a minimum monthly payment rate.

The following variables contain values as described below:

1. `balance` - the outstanding balance on the credit card
2. `annualInterestRate` - annual interest rate as a decimal

The program should print out one line: the lowest monthly payment that will pay off all debt in under 1 year, for example:

```
Lowest Payment: 180
```

Assume that the interest is compounded monthly according to the balance at the end of the month (after the payment for that month is made). The monthly payment must be a multiple of $10 and is the same for all months. Notice that it is possible for the balance to become negative using this payment scheme, which is okay. A summary of the required math is found below:

Monthly interest rate = (Annual interest rate) / 12.0

Monthly unpaid balance = (Previous balance) - (Minimum fixed monthly payment)

Updated balance each month = (Monthly unpaid balance) + (Monthly interest rate x Monthly unpaid balance)

```
monthlyInterestRate = annualInterestRate/12
minimum_fixed = 0
while balance > 0:
   new_balance = balance
   for i in range(12):
       monthly_unpaid = new_balance - minimum_fixed
       new_balance = monthly_unpaid + monthlyInterestRate*monthly_unpaid
       if new_balance < 0:
           break
   minimum_fixed += 10
   if new_balance < 0:
       break

print(minimum_fixed - 10)
```

Problem 3

Problem Set 3

Problem 1 - Check whether a guessed letter in a word

```
secretWord = 'apple'
lettersGuessed = ['e', 'i', 'k', 'p', 'r', 's']

def isWordGuessed(secretWord, lettersGuessed):
    '''
    secretWord: string, the word the user is guessing
    lettersGuessed: list, what letters have been guessed so far
    returns: boolean, True if all the letters of secretWord are in lettersGuessed;
      False otherwise
    '''
    for char in lettersGuessed:
        if char in secretWord:
            secretWord = secretWord.replace(char,"")
    if not secretWord:
        return True
    else:
        return False

print(isWordGuessed(secretWord, lettersGuessed))
```

Problem 2 - Replace letters that exist in a list with an underscore

```
secretWord = 'apple'
lettersGuessed = ['e', 'i', 'k', 'p', 'r', 's']
```

```python
def getGuessedWord(secretWord, lettersGuessed):
    '''
    secretWord: string, the word the user is guessing
    lettersGuessed: list, what letters have been guessed so far
    returns: string, comprised of letters and underscores that represents
      what letters in secretWord have been guessed so far.
    '''
    unguessed = []
    for char in secretWord:
        if char in lettersGuessed:
            unguessed.append(char)
        else:
            unguessed.append("_")
    return "".join(unguessed)


print(getGuessedWord(secretWord, lettersGuessed))
```

Problem 3 - Available unguessed letters

```python
import string

def getAvailableLetters(lettersGuessed):
    '''
    lettersGuessed: list, what letters have been guessed so far
    returns: string, comprised of letters that represents what letters have not
      yet been guessed.
    '''
    alphabet = string.ascii_lowercase
    for char in lettersGuessed:
        if char in alphabet:
            alphabet = alphabet.replace(char,"")
    return alphabet

print(getAvailableLetters(["a","v"]))
```

Problem 4 - The whole hangman game

```python
import random,string

WORDLIST_FILENAME = "problem_set_3\words.txt" # need to have words.txt file

def loadWords():
    """
    Returns a list of valid words. Words are strings of lowercase letters.

    Depending on the size of the word list, this function may
    take a while to finish.
    """
    print("Loading word list from file...")
    # inFile: file
    inFile = open(WORDLIST_FILENAME, 'r')
    # line: string
    line = inFile.readline()
    # wordlist: list of strings
    wordlist = line.split()
    print("  ", len(wordlist), "words loaded.")
    return wordlist

def chooseWord(wordlist):
    """
    wordlist (list): list of words (strings)

    Returns a word from wordlist at random
    """
    return random.choice(wordlist)

# end of helper code
# -----------------------------------

# Load the list of words into the variable wordlist
# so that it can be accessed from anywhere in the program
wordlist = loadWords()

def isWordGuessed(secretWord, lettersGuessed):
    '''
    secretWord: string, the word the user is guessing
    lettersGuessed: list, what letters have been guessed so far
    returns: boolean, True if all the letters of secretWord are in lettersGuessed;
      False otherwise
    '''
    for char in lettersGuessed:
        if char in secretWord:
            secretWord = secretWord.replace(char,"")
    if not secretWord:
        return True
    else:
        return False


def getGuessedWord(secretWord, lettersGuessed):
    '''
    secretWord: string, the word the user is guessing
    lettersGuessed: list, what letters have been guessed so far
    returns: string, comprised of letters and underscores that represents
      what letters in secretWord have been guessed so far.
    '''
    unguessed = []
    for char in secretWord:
        if char in lettersGuessed:
            unguessed.append(char)
        else:
            unguessed.append("_")
    return "".join(unguessed)


def getAvailableLetters(lettersGuessed):
    '''
    lettersGuessed: list, what letters have been guessed so far
    returns: string, comprised of letters that represents what letters have not
      yet been guessed.
    '''
    alphabet = string.ascii_lowercase
    for char in lettersGuessed:
        if char in alphabet:
            alphabet = alphabet.replace(char,"")
    return alphabet


def hangman(secretWord):
    '''
    secretWord: string, the secret word to guess.

    Starts up an interactive game of Hangman.

    * At the start of the game, let the user know how many
      letters the secretWord contains.

    * Ask the user to supply one guess (i.e. letter) per round.

    * The user should receive feedback immediately after each guess
      about whether their guess appears in the computers word.

    * After each round, you should also display to the user the
      partially guessed word so far, as well as letters that the
      user has not yet guessed.

    Follows the other limitations detailed in the problem write-up.
    '''
    guesses_left = 8
    guessed_list = []
    print("Welcome to the game Hangman!")
    print("I am thinking of a word that is " + str(len(secretWord)) + " letters
long")
    print("-----------")
    while not isWordGuessed(secretWord,guessed_list):
      print("You have " + str(guesses_left) + " guesses left")
      print("Available Letters: " + str(getAvailableLetters(guessed_list)))
      guessed_char = input("Please guess a letter: ")
      if guessed_char in secretWord:
        if guessed_char in getAvailableLetters(guessed_list):
          guessed_list.append(guessed_char)
          print("Good guess: " + str(getGuessedWord(secretWord,guessed_list)))
        else:
          print("Oops! You've already guessed that letter: " +
str(getGuessedWord(secretWord,guessed_list)))
      else:
        if guessed_char not in getAvailableLetters(guessed_list):
          print("Oops! You've already guessed that letter: " +
str(getGuessedWord(secretWord,guessed_list)))
        else:
          print("Oops! That letter is not in my word: " +
str(getGuessedWord(secretWord,guessed_list)))
          guesses_left -= 1
          guessed_list.append(guessed_char)
          if guesses_left == 0:
            break
      print("-----------")
    if isWordGuessed(secretWord,guessed_list):
      return print("Congratulations, you won!")
    else:
      print("-----------")
      return print("Sorry you ran out of guesses. The word was " + secretWord +
".")
# When you've completed your hangman function, uncomment these two lines
# and run this file to test! (hint: you might want to pick your own
# secretWord while you're testing)
secretWord = chooseWord(wordlist).lower()
hangman(secretWord)
```

Problem set 4

Problem 1 - Word scores

```python
def getWordScore(word, n):
    """
    Returns the score for a word. Assumes the word is a valid word.

    The score for a word is the sum of the points for letters in the
    word, multiplied by the length of the word, PLUS 50 points if all n
    letters are used on the first turn.

    Letters are scored as in Scrabble; A is worth 1, B is worth 3, C is
    worth 3, D is worth 2, E is worth 1, and so on (see SCRABBLE_LETTER_VALUES)

    word: string (lowercase letters)
    n: integer (HAND_SIZE; i.e., hand size required for additional points)
    returns: int >= 0
    """
    total_score = 0
    for char in word:
        total_score += SCRABBLE_LETTER_VALUES[char]
    total_score *= len(word)
    if len(word) == n:
        total_score += 50
    return total_score
```

Problem 2 - Update hand

```python
def updateHand(hand, word):
    """
    Assumes that 'hand' has all the letters in word.
    In other words, this assumes that however many times
    a letter appears in 'word', 'hand' has at least as
    many of that letter in it.

    Updates the hand: uses up the letters in the given word
    and returns the new hand, without those letters in it.

    Has no side effects: does not modify hand.

    word: string
    hand: dictionary (string -> int)
    returns: dictionary (string -> int)
    """
    copied_hand = hand.copy()
    for char in word:
        copied_hand[char] -= 1
    return copied_hand
```

Problem 3 - Is valid word

```python
def isValidWord(word, hand, wordList):
    """
    Returns True if word is in the wordList and is entirely
    composed of letters in the hand. Otherwise, returns False.

    Does not mutate hand or wordList.

    word: string
    hand: dictionary (string -> int)
    wordList: list of lowercase strings
    """
    copied_hand = hand.copy()
    inHand = True
    for char in word:
        if char not in copied_hand.keys():
            inHand = False
            break
        else:
            copied_hand[char] -= 1
            if copied_hand[char] < 0:
                inHand = False
                break
    return inHand and (word in wordList)
```

Problem 4 - Calculate hand length

```python
def calculateHandlen(hand):
    """
    Returns the length (number of letters) in the current hand.

    hand: dictionary (string-> int)
    returns: integer
    """
    return sum(hand.values())
```

Problem 5 - Play hand

```
def playHand(hand, wordList, n):
    """
    Allows the user to play the given hand, as follows:

    * The hand is displayed.
    * The user may input a word or a single period (the string ".")
      to indicate they're done playing
    * Invalid words are rejected, and a message is displayed asking
      the user to choose another word until they enter a valid word or "."
    * When a valid word is entered, it uses up letters from the hand.
    * After every valid word: the score for that word is displayed,
      the remaining letters in the hand are displayed, and the user
      is asked to input another word.
    * The sum of the word scores is displayed when the hand finishes.
    * The hand finishes when there are no more unused letters or the user
      inputs a "."

      hand: dictionary (string -> int)
      wordList: list of lowercase strings
      n: integer (HAND_SIZE; i.e., hand size required for additional points)

    """
    # BEGIN PSEUDOCODE <-- Remove this comment when you code this function; do your
coding within the pseudocode (leaving those comments in-place!)
    # Keep track of the total score
    total_score = 0
    # As long as there are still letters left in the hand:
    while calculateHandlen(hand) > 0:
        # Display the hand
        print("Current Hand: ",end="")
        displayHand(hand)
        # Ask user for input
        entered_word = input('Enter word, or a "." to indicate that you are
finished: ')
        # If the input is a single period:
        if entered_word == '.':
            # End the game (break out of the loop)
            break
        # Otherwise (the input is not a single period):
        else:
            # If the word is not valid:
            if not isValidWord(entered_word,hand,wordList):
                # Reject invalid word (print a message followed by a blank line)
                print("Invalid word, please try again.\n")
            # Otherwise (the word is valid):
```

```
            else:
                total_score += getWordScore(entered_word,n)
                # Tell the user how many points the word earned, and the updated
total score, in one line followed by a blank line
                print('"'+entered_word+'"'+ " earned " +
str(getWordScore(entered_word,n)) + " points. Total: "+str(total_score)+"
points\n")
                # Update the hand
                hand = updateHand(hand,entered_word)
    # Game is over (user entered a '.' or ran out of letters), so tell user the
total score
    if entered_word == '.':
        print("Goodbye! Total score: " + str(total_score) + " points.")
    else:
        print("Run out of letters. Total score: " + str(total_score) + " points.")
```

Problem 6 - Full Scrabble game

```python
# The 6.00 Word Game
import random

VOWELS = 'aeiou'
CONSONANTS = 'bcdfghjklmnpqrstvwxyz'
HAND_SIZE = 7

SCRABBLE_LETTER_VALUES = {
    'a': 1, 'b': 3, 'c': 3, 'd': 2, 'e': 1, 'f': 4, 'g': 2, 'h': 4, 'i': 1, 'j': 8,
'k': 5, 'l': 1, 'm': 3, 'n': 1, 'o': 1, 'p': 3, 'q': 10, 'r': 1, 's': 1, 't': 1,
'u': 1, 'v': 4, 'w': 4, 'x': 8, 'y': 4, 'z': 10
}

# -----------------------------------
# Helper code
# (you don't need to understand this helper code)

WORDLIST_FILENAME = "problem_set_4/words.txt"

def loadWords():
    """
    Returns a list of valid words. Words are strings of lowercase letters.

    Depending on the size of the word list, this function may
    take a while to finish.
    """
    print("Loading word list from file...")
    # inFile: file
    inFile = open(WORDLIST_FILENAME, 'r')
    # wordList: list of strings
    wordList = []
    for line in inFile:
        wordList.append(line.strip().lower())
    print("  ", len(wordList), "words loaded.")
    return wordList

def getFrequencyDict(sequence):
    """
    Returns a dictionary where the keys are elements of the sequence
    and the values are integer counts, for the number of times that
    an element is repeated in the sequence.

    sequence: string or list
    return: dictionary
    """
    # freqs: dictionary (element_type -> int)
    freq = {}
    for x in sequence:
        freq[x] = freq.get(x,0) + 1
    return freq


# (end of helper code)
# -----------------------------------

#
# Problem #1: Scoring a word
#
def getWordScore(word, n):
    """
    Returns the score for a word. Assumes the word is a valid word.

    The score for a word is the sum of the points for letters in the
    word, multiplied by the length of the word, PLUS 50 points if all n
    letters are used on the first turn.

    Letters are scored as in Scrabble; A is worth 1, B is worth 3, C is
    worth 3, D is worth 2, E is worth 1, and so on (see SCRABBLE_LETTER_VALUES)

    word: string (lowercase letters)
    n: integer (HAND_SIZE; i.e., hand size required for additional points)
    returns: int >= 0
    """
    total_score = 0
    for char in word:
        total_score += SCRABBLE_LETTER_VALUES[char]
    total_score *= len(word)
    if len(word)==n:
        total_score += 50
    return total_score




#
# Problem #2: Make sure you understand how this function works and what it does!
#
def displayHand(hand):
    """
    Displays the letters currently in the hand.

    For example:
    >>> displayHand({'a':1, 'x':2, 'l':3, 'e':1})
    Should print out something like:
       a x x l l l e
    The order of the letters is unimportant.

    hand: dictionary (string -> int)
    """
    for letter in hand.keys():
        for j in range(hand[letter]):
             print(letter,end=" ")       # print all on the same line
    print()                             # print an empty line

#
# Problem #2: Make sure you understand how this function works and what it does!
#
def dealHand(n):
    """
    Returns a random hand containing n lowercase letters.
    At least n/3 the letters in the hand should be VOWELS.

    Hands are represented as dictionaries. The keys are
    letters and the values are the number of times the
    particular letter is repeated in that hand.

    n: int >= 0
    returns: dictionary (string -> int)
    """
    hand={}
    numVowels = n // 3

    for i in range(numVowels):
        x = VOWELS[random.randrange(0,len(VOWELS))]
        hand[x] = hand.get(x, 0) + 1

    for i in range(numVowels, n):
        x = CONSONANTS[random.randrange(0,len(CONSONANTS))]
        hand[x] = hand.get(x, 0) + 1

    return hand

#
# Problem #2: Update a hand by removing letters
#
def updateHand(hand, word):
    """
    Assumes that 'hand' has all the letters in word.
    In other words, this assumes that however many times
    a letter appears in 'word', 'hand' has at least as
    many of that letter in it.
    Updates the hand: uses up the letters in the given word
    and returns the new hand, without those letters in it.

    Has no side effects: does not modify hand.

    word: string
    hand: dictionary (string -> int)
    returns: dictionary (string -> int)
    """
    copied_hand = hand.copy()
    for char in word:
        copied_hand[char] -= 1
    return copied_hand



#
# Problem #3: Test word validity
#
def isValidWord(word, hand, wordList):
    """
    Returns True if word is in the wordList and is entirely
    composed of letters in the hand. Otherwise, returns False.

    Does not mutate hand or wordList.

    word: string
    hand: dictionary (string -> int)
    wordList: list of lowercase strings
    """
    copied_hand = hand.copy()
    inHand = True
    for char in word:
        if char not in copied_hand.keys():
            inHand = False
            break
        else:
            copied_hand[char] -= 1
            if copied_hand[char] < 0:
                inHand = False
                break
    return inHand and (word in wordList)



#
```

```
# Problem #4: Playing a hand
#

def calculateHandlen(hand):
    """
    Returns the length (number of letters) in the current hand.

    hand: dictionary (string-> int)
    returns: integer
    """
    return sum(hand.values())



def playHand(hand, wordList, n):
    """
    Allows the user to play the given hand, as follows:

    * The hand is displayed.
    * The user may input a word or a single period (the string ".")
      to indicate they're done playing
    * Invalid words are rejected, and a message is displayed asking
      the user to choose another word until they enter a valid word or "."
    * When a valid word is entered, it uses up letters from the hand.
    * After every valid word: the score for that word is displayed,
      the remaining letters in the hand are displayed, and the user
      is asked to input another word.
    * The sum of the word scores is displayed when the hand finishes.
    * The hand finishes when there are no more unused letters or the user
      inputs a "."

      hand: dictionary (string -> int)
      wordList: list of lowercase strings
      n: integer (HAND_SIZE; i.e., hand size required for additional points)

    """
    # BEGIN PSEUDOCODE <-- Remove this comment when you code this function; do your
coding within the pseudocode (leaving those comments in-place!)
    # Keep track of the total score
    total_score = 0
    # As long as there are still letters left in the hand:
    while calculateHandlen(hand) > 0:
        # Display the hand
        print("Current Hand: ",end="")
        displayHand(hand)
        # Ask user for input
        entered_word = input('Enter word, or a "." to indicate that you are
finished: ')
        # If the input is a single period:
        if entered_word == '.':
            # End the game (break out of the loop)
            break
        # Otherwise (the input is not a single period):
        else:
            # If the word is not valid:
            if not isValidWord(entered_word,hand,wordList):
                # Reject invalid word (print a message followed by a blank line)
                print("Invalid word, please try again.\n")
            # Otherwise (the word is valid):
            else:
                total_score += getWordScore(entered_word,n)
                # Tell the user how many points the word earned, and the updated
total score, in one line followed by a blank line
                print('"'+entered_word+'"'+ " earned " +
str(getWordScore(entered_word,n)) + " points. Total: "+str(total_score)+"
points\n")
                # Update the hand
                hand = updateHand(hand,entered_word)
    # Game is over (user entered a '.' or ran out of letters), so tell user the
total score
    if entered_word == '.':
        print("Goodbye! Total score: " + str(total_score) + " points.")
    else:
        print("Run out of letters. Total score: " + str(total_score) + "
points.\n")

#
# Problem #5: Playing a game
#

def playGame(wordList):
    """
    Allow the user to play an arbitrary number of hands.

    1) Asks the user to input 'n' or 'r' or 'e'.
      * If the user inputs 'n', let the user play a new (random) hand.
      * If the user inputs 'r', let the user play the last hand again.
      * If the user inputs 'e', exit the game.
      * If the user inputs anything else, tell them their input was invalid.

    2) When done playing the hand, repeat from step 1
    """
    user_choice = 'n'
    played_game_before = False
    while user_choice != 'e':
        user_choice = input("Enter n to deal a new hand, r to replay the last hand, 
or e to end game: ")
        if user_choice == 'n':
            # play a new game
            hand = dealHand(HAND_SIZE)
            playHand(hand,wordList,HAND_SIZE)
            played_game_before = True
        elif user_choice == 'r':
            # if played before
            if played_game_before:
                # print start a new game
                playHand(hand,wordList,HAND_SIZE)
            # else (not played before)
            else:
                # print not played before
                print("You have not played a hand yet. Please play a new hand 
first!")
        elif user_choice != 'e':
            print("Invalid command.")






#
# Build data structures used for entire session and play game
#
if __name__ == '__main__':
    wordList = loadWords()
    playGame(wordList)
```

Problem 7 - Computer playing scrabble

```
def playGame(wordList):
    """
    Allow the user to play an arbitrary number of hands.

    1) Asks the user to input 'n' or 'r' or 'e'.
        * If the user inputs 'e', immediately exit the game.
        * If the user inputs anything that's not 'n', 'r', or 'e', keep asking them
again.

    2) Asks the user to input a 'u' or a 'c'.
        * If the user inputs anything that's not 'c' or 'u', keep asking them
again.

    3) Switch functionality based on the above choices:
        * If the user inputted 'n', play a new (random) hand.
        * Else, if the user inputted 'r', play the last hand again.

        * If the user inputted 'u', let the user play the game
          with the selected hand, using playHand.
        * If the user inputted 'c', let the computer play the
          game with the selected hand, using compPlayHand.

    4) After the computer or user has played the hand, repeat from step 1

    wordList: list (string)
    """
    user_choice_start = 'n'
    played_game_before = False
    while user_choice_start != 'e':
        user_choice_start = input("Enter n to deal a new hand, r to replay the last
hand, or e to end game: ")
        if user_choice_start == 'n':
            while True:
                user_choice_play = input("Enter u to have yourself play, c to have
the computer play: ")
                if user_choice_play == 'c':
                    break
                elif user_choice_play == 'u':
                    break
                else:
                    print("Invalid command.")
            if user_choice_play == 'u':
                # play a human new game
                hand = dealHand(HAND_SIZE)
                playHand(hand,wordList,HAND_SIZE)
                played_game_before = True
            elif user_choice_play == 'c':
                # play a computer new game
                hand = dealHand(HAND_SIZE)
                compPlayHand(hand,wordList,HAND_SIZE)
                played_game_before = True

        elif user_choice_start == 'r':
            # if played before
            if played_game_before:
                while True:
                    user_choice_play = input("Enter u to have yourself play, c to
have the computer play: ")
                    if user_choice_play == 'c':
                        break
                    elif user_choice_play == 'u':
                        break
                    else:
                        print("Invalid command.")
                if user_choice_play == 'u':
                # play a human new game
                    playHand(hand,wordList,HAND_SIZE)
                elif user_choice_play == 'c':
                    # play a computer new game
                    compPlayHand(hand,wordList,HAND_SIZE)
                else:
                    print("Invalid command.")
                    # print start a new game
            # else (not played before)
            else:
                # print not played before
                print("You have not played a hand yet. Please play a new hand
first!")
        elif user_choice_start != 'e':
            print("Invalid command.")
```

Debugging and Exception Handling

Debugging

Input method

If you put "*input*" at the end of an "*If*" statement or a "*For*" loop, it would **pause the iteration** until you press spacebar.
Useful if you have problems with your code and you don't get your desired output.

```
input()
```

Print method

Adding "*print*" at the end of an "*If*" statement or a "*For*" loop, will print the variable and help you debug the code.

```
print('output:', output)
```

Exception Handling

Using try

Using raise

```
raise <exceptionName> (<arguments>)
raise ValueError("Something is wrong")
```

Using assert

```
def avg(grades):
    """
    input: a list of grades
    output: an average of the grades

    if there's an empty list, print "no grades data"
    resulting exception is an "AssertionError"
    Typically used to check inputs.
    """
    assert not (len(grades) == 0), 'no grades data'
    return sum(grades)/len(grades)
```