**Python Core - Exercises**

When attempting these questions, make sure to write all code from scratch.

You will learn faster if you avoid copying and pasting.

**1. Introductory Exercises**

**Exercise 1 (Strings):**

- Create a string with the text 'Hello World 111'. Assign to my\_string.

- Try to convert the string to an integer using int(my\_string). What happens?

- Grab the first five characters of the string using slicing. \* Google this for now. We will learn about slicing later.

**Exercise 2 (Functions and Conditionals):**

- Create a function 'my\_func' that accepts one integer argument 'x'.

- If 'x' is less than 5, print out 'x is less than 5'.

- Otherwise, print out 'x is greater than 5'.

**Exercise 3 (Lists):**

- What methods are supported for lists?

- Create a list with elements 'a', 'b' and 'c'.

- Add 'd' to the end of this list.

- Get the first two elements of the list.

**Exercise 4 (Sets):**

- Create a set containing 1,2,3 and another set containing 4,5,6

- Use the intersection method to find the intersection between the two sets

- Find the union of the two sets

**Exercise 5 (Iteration and Conditionals):**

- Create a list containing 1,2,3,4,5,6

- Loop through the list. Print out the value if it is an even number

**Exercise 6 (File IO):**

- Create a file containing text '<Your name here> is awesome!'

- Read in this file and print out its contents

- Append to the file 'I like muffins'

- Append to the file 'I love learning!'

- Check that the contents have been appended appropriately

**Exercise (Exception Handling):**

- Create a function to read and print out the first line of a file

- If the file does not exist, print out 'My file is missing...:('

**2. Objects and Types**

**Exercise 1 - List basics:**

- Create a list my\_list containing elements 'a', 'b', 'c'.

- Create a copy of my\_list.

- Verify that a copy has been created.

**Exercise 2 - Strings:**

- Create a string my\_string1 containing 'i am an app. '

- Create a string my\_string2 containing ' monkzey crayonszone love'

- Strip all leading and trailing whitespace from my\_string1 and my\_string2.

- In my\_string2, replace all z's with spaces.

- Concatenate my\_string1 and my\_string2. Assign this to my\_string3

- Get the length of my\_string3

- Use slicing to get every fourth character in the my\_string3.

- If your answer looks good, then the answer is probably right.

**Exercise 3 - List iteration:**

- Create list called evens containing [0,2,4,6,8]

- Create list called odds containing [1,3,5,7,9]

- Create a new empty list called mangled.

- In a loop:

- Append to mangled the first element of odds

- Append to mangled the first element of evens

- Append to mangled the second element of odds

- Append to mangled the second element of evens

- Repeat the process until both lists are exhausted.

- Note the resulting mangled list should look like [1,0,3,2,5,4,7,6,9,8]

**Exercise 4 - Dictionaries:**

- Create a dictionary with the following keys and values:

key <your name here>

value <your favourite food>

eg. {'Albert':'Okonomiyaki'}, {'Lawrence':'Beer'}, {'David':'Vegetables'}

- Repeat the above for five of your classmates

- Create a list containing all dictionaries above

- Create another list containing only dictionaries with people with names starting with the same character as your first name.

**Exercise 5 - Dictionaries (again):**

- You have the following dictionaries {'a':1, 'b':2}, {'c':3}, {'d':4, 'e':5, 'f':6}

- Write a function that concatenates all these into one dictionary.

**Exercise 6 - Mutability**

- Create an empty lists and call this my\_list1

- Create a reference to my\_list1. Call this my\_list2. (ie. run my\_list2 = my\_list1)

- Append 'hello' to my\_list1.

- Does my\_list2 change? Why or why not?

**Exercise 7 - Vowel Counter**

- Create a function that returns the vowel count for a text document.

- For example, if a file contains the following text:

'''

hello

data engineers

'''

the function should return {'a':2, 'e':4, 'i':1, 'o':1, 'u':0}. Try to use only concepts that we have already covered.

- Note to get the length of a sequence my\_seq, we can use len(my\_seq)

**\* Exercise 8 - Mutability**

- Create a list my\_list containing ['a','b','c']

- Create two lists of lists lol1 and lol2 containing [my\_list, my\_list, my\_list]

- Set the first element of the first list in lol1 to 'Z'.

ie. lol1[0][0] = 'Z'

- What happens to my\_list? What happens to lol1 and lol2? Why?

**\* Exercise 9 - Mutability**

- Create a list my\_list1 containing ['a','b','c']

- Create a list my\_list2 = my\_list1\*1

- What does my\_list2 contain?

- Run my\_list2[0] = 'Z'. What happens to my\_list1? Why?

**3. Program Structure**

**Exercise 1**

- Define a variable 'my\_name'. Assign your name to the variable.

- Using an if statement:

- print out 'my name is short' if the total number of characters in your name is less than 10.

- print out 'my name is long' if the total number of characters in your name is greater than 17.

- otherwise, print out "my name is about average length"

- try assigning other values to my\_name, to check that your code is working as expected.

**Exercise 2**

- Define a variable 'my\_list'. Set it to be a list of all files in your C directory.

Hint: using os.listdir('C:\\') will be helpful.

- Create a new list containing everything in my\_list that does not contain spaces.

**Exercise 3**

- Create a variable 'random\_no'. Set it to a random variable between 1 and 20 inclusive.  
You can use random.randint(1,20).

- While the number is not equal to 20, assign 'random\_no' to a new random number.  
Keep repeating this process until 'random\_no' is equal to 20.

- Print out the number of draws made until 'random\_no' was equal to 20.

- Use a 'for' loop to repeat the above steps 100 times,

and report the mean (average) number of draws made until 'random\_no' was equal to 20.

**Exercise 4**

- Read in a file that does not exist, catch the exception, and report the type of exception.

**Exercise 5**

- Consider the following mathematical operations. Which of these will raise an exception, and of what type? For those that don't raise an exception, what is the result?

- dividing one by zero

- taking the logarithm of zero

- taking the square-root of minus one

**Exercise 6**

- Consider the python objects my\_list = [0]; my\_tuple = (0,);my\_int = 0.  
What are the Boolean values of these objects?

- Try removing the brackets from the definition of my\_tuple -- is the tuple still defined correctly?

- Can you print out the Boolean value of these objects using only the name of the object and the 'not' operator?

**\* Exercise 7**

- Create a function to do element-wise division. Call this function elementwise\_div(numerators, denominators). The arguments numerators and denominators should be lists.

- For example, elementwise\_div([1,2,6],[2,2,3]) should return [0.5,1,6].

- If the element-wise division cannot be computed, then return None instead.  
For example, elementwise\_div([1,2,'3'],[1,0,3]) should return [1,None,None].

**\* Exercise 8**

- Write a function called compare\_numbers, taking two parameters, that prints out a string describing whether two numbers have the same sign, and whether they have the same magnitude.

For example, when this function is called with the following parameters:

compare\_numbers(-1,-1)

compare\_numbers(-2,-1)

compare\_numbers(1,-1)

compare\_numbers(-1,3)

Then this will result in the following output:

-1 and -1 have the same sign, and the same magnitude

-2 and -1 have the same sign, but different magnitudes

1 and -1 have different signs, but the same magnitude

-1 and 3 have different signs, and different magnitudes

**4. Functions**

**Exercise 1 - Basic function**

- Create a function that calculates the momentum of an object given the mass and velocity.

- Set the default value of m to be 1 (kg).

**Exercise 2 - Using \***

- Write a function accepts a variable number of string arguments (without keywords).

The function is to print out every argument separated by a new line.

- For example, my\_func('hello', 'oh-no', 'tumbleweed') should print out:

'hello'

'oh-no'

'tumbleweed'

**Exercise 3 - \*args and \*\*kwargs**

- Write a function to print out all keyword arguments that begin with a letter alphabetically before 'p'.

**Exercise 4 - Scoping**

- Explain the scoping rules for a function to the person or wall on your left.

- Write code to illustrate what you just explained.

**Exercise 5 - Decorators**

- What is the purpose of a decorator?

- Write a decorator to print 'function finished!' after the wrapped function is complete.

**Exercise 6 - Comprehensions / maps**

- Create a numerical list [1,3,5,2,4]. Write a list comprehension to iterate through the list.

- Do the same thing with a generator expression.

- When would we prefer to use a generator expression?

- Repeat the same thing with the map function with a lambda function.

**Exercise 7 - Dictionary Comprehensions**

- Create two lists ['a','b','c'], [1,2,3].

- Using a comprehension, create a dictionary containing {'a':1, 'b':2, 'c':3}

**\* Exercise 8 - Odd numbers intersection**

- Create a function called odds\_intersection(list1, list2).

The function should return a collection of odd numbers that list1 and list2 have in common.

For example, odds\_intersection(['a','b',1,2,'3',4,5,'c'], ['z','y,'x',3,2,1,9,7,5,5] should return [1, 5].

**\* Exercise 9 - kwargs decorator**

- Create a decorator called kwargs\_only to force the user to only use keyword arguments with a function.

- For example, suppose we had a function decorated with kwargs\_only

@kwargs\_only

def my\_function(x):

return x

- Running my\_function(5) should return an error, but running my\_function(x=5) should work.

**\* Exercise 10 - Custom map**

- Create a function my\_map(f, my\_list) that returns a generator that applies f to every element in my\_list.

This behaviour will be similar to that of the in-built map function.

**\* Exercise 11 -- transpose a list of lists**

- consider the following list of lists: lol = [[1,2,3],[4,5,6],[7,8,9]].

- How can we transpose it, so that the rows and columns are reversed,

i.e. we get [[1, 4, 7], [2, 5, 8], [3, 6, 9]] ?

- Hint: you can use the '\*' operator and 'zip()' function to obtain a correct list of tuples,

and then the 'map()' function to turn this into a list of lists.

**5. Classes and OOP**

**Exercise 1 - Guessing Game**

Create a class to play a guessing game:

At the start of the game, a random number between 0 and 50 is chosen.

The player has N attempts to guess this number.

After a guess, several things can happen:

- The player gets the number right and wins. The game terminates.

- The player gets the number wrong. The number of attempts is decremented.

- If the number of attempts is zero, the player loses and the game terminates.

- Otherwise, the game tells the player whether the true number is higher or lower than the guess.

**Exercise 2 - Bank accounts**

- Create a class called StandardAccount. The class should be initialized with:

- balance: the starting balance

- interest rate: the interest

The class should implement the following:

- withdraw(amount):   
Reduces the balance in the account by the amount plus transaction fee. Returns the amount.

- deposit(amount):   
Adds the amount to the balance.

- calculate\_interest():   
Adds/subtracts the interest from the current balance.

- Create a class called BadAccount. The class should be the same as the StandardAccount, except that it is initialized with a transaction\_fee. For example, BadAccount(100, 0.01) initializes a bank account with a starting balance of 100 and a transaction fee of 1% per transaction. Every time a user withdraws from the BadAccount, a transaction fee proportional to the amount is taken.

- Create a class called PremierAccount. The class should be the same as the StandardAccount, except that it is initialized with min\_amount. For example, PremierAccount(100, 5) initializes a bank account with a starting blaance of 100 and a min\_amount of 5. When the user withdraws over $min\_amount in a transaction, they get a point. Every time they accumulate 5 points, $5 is added to the account balance.

- Implement the appropriate error checks.

**Exercise 2b - Bank accounts Again**

Create 5 instances of bank accounts and store them in a list.

For each bank account, call the accrue interest method.

**Exercise 3 - Naughts and Crosses**

Implement a class to play naughts and crosses.

**Exercise 4 - Hangman**

Implement a class to play hangman.

**Exercise 5**

Create a class called CSVReader. The class is intended to allow the user to get data from a csv file.

- Implement the \_\_init\_\_(self, file\_path) method which:

- Stores the path to the file

- Initializes the 'columns' attribute to None.

- Initializes the 'data' attribute to None.

- Implement the read(self) function which:

- Reads the first line of the file and stores it into self.columns as a list.

For example, if the first line contains "id, gender, occupation", then self.columns should contain:

['id', 'gender', 'occupation']

- Reads all other lines of the file and stores them in self.data as a list of lists. Using example data, our data should contain:

[['1', 'm', 'brick layer'],

['2', 'f', 'window cleaner'],

['3', 'm', 'swimmer'],

['4', 'f', 'astronaut']]

- Implement the get(self, column\_name, row\_number) function that returns the value for the given row and columns. For example, using the example data:

reader.get('occupation', 0) # returns brick layer

reader.get('gender', 1) # returns f

# Example Data:

id, gender, occupation

1, m, brick layer

2, f, window cleaner

3, m, swimmer

4, f, astronaut

**\* Exercise 6 - Connect 4**

Implement a class to play connect 4.