Name: ___________________________  Partner: ___________________________

Python Activity 38: Hash Function

<table>
<thead>
<tr>
<th>Learning Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students will be able to:</td>
</tr>
<tr>
<td><strong>Content:</strong></td>
</tr>
<tr>
<td>• Describe how a <strong>Hash Table algorithm</strong> works.</td>
</tr>
<tr>
<td>• Define a <strong>Hash Function</strong>'s role in dictionaries and sets.</td>
</tr>
<tr>
<td>• List the main requirements of a good Hash Function.</td>
</tr>
<tr>
<td><strong>Process:</strong></td>
</tr>
<tr>
<td>• Write a Hash Function for user-defined types.</td>
</tr>
</tbody>
</table>

**Prior Knowledge**

• Activities 1-28, Dictionaries, Sets

*If you encounter any issues/typos, let Iris know! Questions? Ask Iris or the POGILing forum*

**Concept Model:**

FYI: A python Dictionary is an implementation of a **Hash Table algorithm**. Hash Tables are an array of items which calculate an index from the item's key and use this index to place the data into the array. A **Hash Function** is the function that calculates this index (i.e., a **Hash**) from the item's key.

The following diagram of a Hash Table represents the following python dictionary (a mapping of dog names as keys to their favorite food as values):

```python
dogFood = { 'tally': 'bananas', 'linus': 'everything', 'pixel': 'cheese' }
```

<table>
<thead>
<tr>
<th><strong>Hash Table Algorithm</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keys</strong></td>
</tr>
<tr>
<td>'tally'</td>
</tr>
<tr>
<td>'linus'</td>
</tr>
<tr>
<td>'pixel'</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

If you encounter any issues/typos, let Iris know! Questions? Ask Iris or the POGILing forum.
1. Where are the Hash Table keys located in this diagram?
________________________________________________________________________
________________________________________________________________________

2. If we wanted to look for the value mapped to 'tally' what Hash/index would we look for?
________________________________________________________________________

3. What is stored in the bucket located at index 7104?
________________________________________________________________________

4. What appears to be stored in the Buckets in the Hash Table diagram?
________________________________________________________________________
________________________________________________________________________

Let's add another key-value mapping to our dictionary:

dogFood['wally'] = 'carrots'

5. Let's assume the Hash for 'wally' is 7104. Why might this be a problem?
________________________________________________________________________
________________________________________________________________________

One option to resolve this issue is to link the new data with the overlapping hash to the previous data at that same Hash index, like in the diagram below:

6. In this example, how many key-value pairs are stored at Hash 7104? ____________

FYI: For a given value, its Hash will always be the same, although it may change between successive python sessions.
7. If we want to print ('dogFood['wally']') python has to first retrieve the Hash for 'wally', then look at that index in the Hash Table, then iterate through all the items in the buckets located at that hash until it finds the 'wally' key. How many items does python have to look at before it find the 'wally' key? ______________

8. If we add dogFood['annie']='hamburger' to our dictionary, and the Hash for 'annie' is 3998, what are the steps necessary to print ('dogFood['annie']')?
________________________________________________________________________
________________________________________________________________________

Critical Thinking Questions:

FYI: In python, hash(obj) is a built-in function that produces an integer value that represents the Hash for obj.

1. Examine the sample code below from interactive python:

```python
0 >>> s = 'hello world'
1 >>> s2 = 'hello world'
2 >>> s is s2
3 False
4 >>> hash(s)
5 4963799451833479185
6 >>> hash(s2)
7 4963799451833479185
```

a. What would >>> s == s2 output? _____________________________

b. Are s and s2 the same object instance, how do you know?

________________________________________________________

c. What does the value outputted on line 5 represent?

________________________________________________________

d. What does the value outputted on line 7 represent?

________________________________________________________

e. What would >>> hash(s) == hash(s2) output? _____________________________

f. What does this tell us about objects that share a value, but are different instances?

________________________________________________________

g. What might >>> hash('hello world') output?

________________________________________________________
2. Examine the sample code below from interactive python:

```
Interactive Python
0 >>> s = 'hello world'
1 >>> t = s + '!
2 >>> hash(s)
3 4963799451833479185
4 hash(t)
5 -877405096577060213
```

a. How does t differ from s? How many characters differ?

______________________
________

b. How does hash(t) differ from hash(s)? How many digits differ?

______________________
________

c. What does this tell us about the Hashes generated for objects that are very similar?

______________________

3. Examine the sample code below from the Terminal:

```
Terminal
0 -> python3
1 >>> s = 'hello world'
2 >>> hash(s)
3 4963799451833479185
4 -> exit() # exits interactive python
5 -> python3 # starts a new session of interactive python
6 >>> s = 'hello world'
7 >>> hash(s)
8 4686556288558268365
```

a. How does s on line 1 differ from s on line 6?

______________________

b. How does the value from hash(s) on line 2 differ from the value from hash(s) on line 7?

______________________

c. What happens between lines 2 and 7 that might explain this?

______________________

d. What does your answer in 3c tell us about the Hashes generated for values across different python sessions?

______________________
4. Examine the sample code below from interactive python:

```
0 >>> hash(1)
1 1
2 >>> hash(2)
3 2
4 >>> hash(10000000000000000000)
5 10000000000000000000
6 >>> hash(10000000000000000000)
7 77627963145224196
```

a. How do the arguments and values returned on lines 0 & 1, and 2 & 3 differ?

b. What might be the output for \( \text{hash}(3) \)?

c. What does this say about Hashes for integers?

d. How do the arguments and values returned on lines 4 & 5 differ?

e. How do the arguments and values returned on lines 6 and 7 differ?

f. What does this say about Hashes for large integers?

5. Examine the sample code below from interactive python:

```
0 >>> class Flower:
1 ... 
2 ...   slots = ['sepals', 'petals']
3 ...   def __hash__(self):
4 ...     return self.petals + self.sepals
5 >>> rose = Flower()
6 >>> rose.petals = 10
7 >>> rose.sepals = 5
8 >>> hash(rose)
9 15
```

a. If we were to call \texttt{rose.__hash__()} (\textit{underscore guilt}!), what would the output be?

b. How does the output from \texttt{rose.__hash__()} differ from the output of \texttt{hash(rose)} on line 7?

c. What special method is likely being called on line 7?
d. Write a few lines of python to generate another Flower with the same Hash as rose using different values for petals and sepals:

```
_________________
______
_________________
__________
__
_________________
______
_________________
__________
__
_________________
______
_________________
__________
__
```

e. How common might the Hash value for rose be?

```
_________________
______
_________________
__________
__
_________________
______
_________________
__________
__
_________________
______
_________________
__________
__
```

f. If it's super common for different flowers to have the same sum of petals & sepals, how might we modify Flower's hash function to have fewer collisions?

```
_________________
______
_________________
__________
__
_________________
______
_________________
__________
__
_________________
______
_________________
__________
__
```

g. Write a few lines of python to make a better __hash__ method for Flower:

```
_________________
______
_________________
__________
__
_________________
______
_________________
__________
__
_________________
______
_________________
__________
__
```

**FYI:** A good Hash Function produces equivalently sized integers for values evenly distributed to decrease collisions. However, if there are no collisions with a large dataset, this may be inefficient.

Application Questions: Use the Python Interpreter to check your work

1. Write a __hash__ (self) method for the Course class which represents a college course with a department prefix (_dept), course number (_num) and target GPA average (_gpa) as slots:

   ```python
   class Course:
       __slots__ = ['_dept', '_num', '_gpa']
   def __hash__(self):
       __________________________
       __________________________
       __________________________
       __________________________
       __________________________
       __________________________
   ```

   Will your Hash Function produce integers?

   Will your Hash Function produce the same value for objects with the same value?
Will your Hash Function produce evenly distributed values for Course objects?

2. Write a `__hash__` method for the Automobile class which represents a car with a make (i.e., brand like Tesla) and a model (like model S):

```python
class Automobile:
    __slots__ = ['_make', '_model']
def __hash__(self):
    # Your implementation here
```

Will your Hash Function produce integers?

Will your Hash Function produce the same value for objects with the same value?

Will your Hash Function produce evenly distributed values for Automobile objects?