Name:_____ **Partner:** _____

Python Activity 61: Java – More Data Types

Java is *programming language* that shares some commonalities with Python. Understanding the differences between Java and Python *data types* help us better understand the Python data structures that we know and love!

Learning Objectives
Students will be able to:
Content:
- Compare and contrast Python & Java data types
- Describe the use of Java ***Strings, Arrays, ArrayLists,*** and ***Hashmaps***

Process:
- Write Java code equivalents of Python code using Strings, Arrays, ArrayLists, and Hashmaps

Prior Knowledge
- Python concepts: Python, Java

Critical Thinking Questions:

STRING

1. Match the Python code on the left with what you think is the Java equivalent on the right:

```
astr = "Pixel"
astr[3]
str[2:3]
Indexing
len(astr)
astr.find('x')
astr.split()
astr.split(',')
astr + astr
astr.strip()
Slicing operator
Immutable
astr.upper()
astr.lower()
str(astr.split())
```

```
String astr = "Pixel";
astr.length();
Method calls
astr.indexOf('x');
astr.concat(astr);
astr.toUpperCase();
astr.toLowerCase();
astr.charAt(3);
Immutable
astr.substring(2, 3);
astr.trim();
astr.split(",");
astr.split(" ");
Substrings through methods
Arrays.toString(astr.split(" "));
```

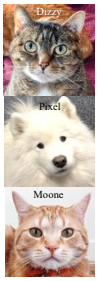
FYI: ***Strings in Java*** are immutable like in Python, but do not support the [ ] slicing operator!

ARRAYLIST

2. Match the Python code on the left with what you think is the Java equivalent on the right:

```
alist = []
alist.append("Moone")
alist.append("Pixel")
print(alist)
alist[1]
alist[1] = "Dizzy"
```

```
ArrayList<String> alist = new ArrayList<String>();
alist.get(1);
alist.set(1, "Dizzy");
System.out.println(alist);
alist.add("Moone");
alist.add("Pixel");
```



FYI: ***ArrayLists in Java*** are roughly equivalent to Lists in Python: they're both dynamic arrays that grow & shrink in size automatically. However, Java ArrayLists cannot use the [ ] operator, nor can they contain heterogeneous *types* of objects, and they require that you declare what type of object is contained in the ArrayList.



ARRAY

3. a. Observe the following Java code (on the left) and its associated output (on the right):

Java code on the left, output on the right

```
1 import java.util.Arrays;
2 String[] myList = new String[] {"Pixel", "Dizzy", "Moone"};
3 System.out.println(Arrays.toString(myList));
                                                    [Pixel, Dizzy, Moone]

4 System.out.println(myList[2]);                    Moone
5 myList[0] = "Pickles";
6 System.out.println(Arrays.toString(myList));      [Pickles, Dizzy, Moone]
```

Underline where you see the class, `Arrays`, being referenced. Why might using this class be necessary? _____

What might each of the lines of Java code do?

1 _____ 4 _____

2 _____ 5 _____

3 _____ 6 _____

FYI: An ***array*** is a primitive data type in Java that is somewhat similar to Python Lists. Arrays support the [ ] operator, but they cannot dynamically shrink and grow: you need to specify its initial size, or its values at declaration. Much like ArrayLists, arrays require you to specify what type of object is stored in the array and hetereogeneous types of objects cannot be stored in them.

b. Observe the following Java code below:

Java example

```
1 import java.util.Arrays;
2 String[] myList = new String[3];
3 myList[0] = "Pixel";
4 myList[1] = "Dizzy";
5 myList[2] = "Moone";
6 System.out.println(Arrays.toString(myList));
```

Underline the code that is different from the previous example.

What will be the output of line 6? _____

Do we have an equivalent to line 2 from this example in part (a)? Why might that be?

_____

FYI: Because Java is ***compiled*** rather than interpreted, there is no "interactive Java." However, for the sake of comparing to python, we'll present Java below in an interactive-like format.

HASHMAP

4. Below, the Python code on the left accomplishes the same as the Java code on the right:

```
>>> cs_courses = dict()

>>> cs_courses[134] = "Intro"
>>> cs_courses[136] = "Data Struct"
>>> cs_courses[256] = "Algorithms"
>>> cs_courses[256]
'Algorithms'
>>> cs_courses[134]
'Intro'
>>> 134 in cs_course
True
>>> 361 in cs_courses
False
>>> "Data Struct" in cs_courses.values()
True
```

```
HashMap<Integer, String> csCourses;
csCourses = new HashMap<Integer, String>();
csCourses.put(134, "Intro");
csCourses.put(136, "Data Struct");
csCourses.put(256, "Algorithms");
csCourses.get(256);
Algorithms
csCourses.get(134);
Intro
csCourses.containsKey(134);
true
csCourses.containsKey(361);
false
csCourses.containsValue("Data Struct");
true
```

a. What do each of the lines of Java code do?

```
HashMap<Integer, String> csCourses;
csCourses = new HashMap<Integer, String>();
```

_____

```
csCourses.put(134, "Intro");
```

_____

```
csCourses.get(256);
```

_____

```
csCourses.containsKey(134);
```

_____

```
csCourses.containsValue("Data Struct");
```

_____

b. Why might the last two lines of Java code differ so much from their Python equivalents?

_____

FYI: ***HashMaps in Java*** are roughly equivalent to dictionaries in Python. Both provide easy access to key, value pairs. Both provide methods for efficiently interacting with the keys, values. And both require keys to be unique. HashMaps do not support the [ ] operator; we must use methods instead.