Name:_____ **Partner:** _____

Python Activity 12: Lists

Holding and accessing collections of objects helps code scale.

Learning Objectives

Students will be able to:

Content:

- Define a **list**
- Identify **elements** of a list
- Explain the purpose of positive and negative **index**es in a list**.**
- Explain how to access individual elements of a list as well as subsequences of the list
- Explain how to find if an item is contained within a list

Process:

- Write code that prints a list, finds the **len**gth of a list, **slice**s a list
- Write code that determines if an item is or is not contained **in** a sequence
- Write code that adds items to a list through **concatenation**

Prior Knowledge

- Variables, string literals, types, conditionals, functions

Concept Model:

Examine the following partially completed code:

Concept Model

```
def print_month(num_month):
    # num_month is a number between 0 & 11, representing Jan - Dec
    str_month = '??'
    # What code needs to go here?
    print("The month is", str_month)
```

CM1. If we wanted the function `print_month` to display a string representation of the numerical month stored in `num_month` (e.g., `print_month(0)` displays `January`, `print_month(3)` displays `April`), summarize what code we would have to write to make this possible, using only concepts we've already learned:

_____

_____

CM2. Will this approach *scale* for larger problems (say, if we wanted a similar mapping between the numerical year 1999 and the string representation, `nineteen ninety-nine`, and *all* other years up to now)?

_____

_____

Critical Thinking Questions:

> **FYI:** A ***sequence*** is an object that stores multiple data items in a contiguous/ordered manner. Two types of sequences are **strings** and **lists.** Each value stored in a list is called an **element.**

1. Examine the sample lists below.

Sample Lists in Python

```
digits = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
fruits = ["apple", "banana", "cantelope", "pear", "orange"]
student_data = ["Jones", 10234, 3.5, "Brown", True, 2.8, 'i']
```

a. How many **elements** does the list named **digits** contain? _____

b. What type of data is stored in each list (String, numeric)?

- **digits** list:

  _____

- **fruits** list:

  _____

- **student_data** list:

  _____

c. How would you define a **list?**

_____

d. Why might a **list** be useful?

_____

_____

2. Many of the operators we know for strings (a *sequence* of characters) are similar for lists (a *sequence* of objects)! As a review of the string operators, draw lines between the left column and the right, matching the sequence operations we have learned that work for **strings**, to the result of those operations:

Operation	**Result**
`seq[i]`	True if `x` is contained within `seq`
`seq[startIncl : endExcl]`	slice of `seq`: `startIncl` to `endExcl` with step `step`
`seq[startIncl : endExcl : step]`	the i'th item of `seq`, when starting with 0
`len(seq)`	slice of `seq` from `startIncl` to `endExcl`
`seq1 + seq2`	False if `x` is contained within `seq`
`x in seq`	length of `seq`
`x not in seq`	The concatenation of `seq1` and `seq2`

3. Knowing these string operators, map the python code on the left to the expected output on the right. Assume `fruits = ["apple", "banana", "cantelope", "pear", "orange"]`

Operation	**Result**
`fruits[1]`	`['apple','banana', 'cantelope', 'pear', 'orange', 'strawberry']`
`fruits[-2]`	`['orange', 'pear', 'cantelope', 'banana', 'apple']`
`fruits[1:4]`	`False`
`fruits[0:5:2]`	`True`
`fruits[::-1]`	`'pear'`
`len(fruits)`	`['banana', 'cantelope', 'pear']`
`fruits + ['strawberry']`	`'banana'`
`'coconut' in fruits`	`['apple', 'cantelope', 'orange']`
`'lemon' not in fruits`	`5`

3. Examine the following program and its output:

Program :

```
legumes = ["beans", "peas"]
vegs = ["asparagus", "broccoli", "carrot"]

combine = legumes + vegs
print(combine)
print(legumes)
```

Output:

```
['beans', 'peas', 'asparagus', 'broccoli', 'carrot']
['beans', 'peas']
```

a. Draw lines between the `print()` statements in the program and their associated output.

b. What is stored in `combine`?

_____

c. At the end of the code, what is stored in `legumes`? How has it changed from the beginning?

d. At the end of the code, what is stored in `vegs`? How has it changed from the beginning?

FYI: The **Concatenation Operator** + allows you to append one sequence, such as Lists or strings, to the end of another sequence of the same type. It returns the *new*, appended sequence.

3. Examine the following program and its output that continues from the previous code:

Program :

```
vegs + ["kale"]
print(vegs)

vegs = vegs + ["beet"]
print(vegs)
```

Output:

```
['asparagus', 'broccoli', 'carrot']
['asparagus', 'broccoli', 'carrot', 'beet']
```

a. What *type* of variable is `vegs`?

What *type* of variable is `["kale"]`?

What *type* of variable is `"kale"`?

Why might we get a `TypeError: can only concatenate list (not "str") to list` if we write `vegs + "kale"` rather than `vegs + ["kale"]`?

b. At the end of the code, what is stored in `vegs`? How has it changed from the beginning?

c. How does the value of `vegs` change after the line `vegs + ["kale"]`?

Why might we have to write `vegs = vegs + ["beet"]` rather than just `vegs + ["kale"]` to update the value stored in `vegs`?

d. Write a *single* line of code that adds the strings "lentil" and "chickpea" to `legumes`.

Application Questions: Use the Python Interpreter to check your work

1. Create a program that prints a given list, prompts the user for a name and average, adds the new information to the list and prints the new list. It should produce output similar to the following:

```
LIST: ['Mary Smith', 132, 'Jean Jones', 156, 'Karen Karter', 167]
Name to add to the list: Ann Kert
Average: 189
UPDATED LIST: ['Mary Smith', 132, 'Jean Jones', 156, 'Karen Karter', 167, 'Ann Kert', 189]
There are now 8 items in the list.
```

2. Revise the previous program so that it allows the user to enter the name of a person and an average, but only if that person does not already exist in the list.

3. Create a function, `extract_palindromes`, that takes a `list` of `str`, `word_list`, as a parameter and returns a list of only the palindromes in that list (words that are the same backwards and forwards). Begin by *decomposing* the problem into smaller steps.