LISTS REVIEW

Lists contain many pieces of data in a single variable. They are accessible by indexes. A list can be identified with square brackets.

```
my_string_list = ["apple", "oranges", "bananas"]
print(my_string_list[0]) # apples
```

LISTS

Lists can contain different data types, and even full lists!

```
my_string_list = ["apple", "oranges", "bananas"]
print(my_string_list[0]) # apples (string)
my_int_list = [2, 3, 10]
print(my_int_list[1]) #3 (integer)
my_float_list = [2.4, 502.4, 2.5]
print(my_float_list[2]) #2.5 (float)
my_list_list = [ [1,4,5], [3,5,4], [4,2,5]]
print(my_list_list[1][1]) #5 (list)
```

LISTS

Lists can contain multiple data types. List is the entire structure (with its own methods) and each item can be accessed and on its own. Its important to pay attention to data types if your list is like this!

```
my_mixed_list = [2.4, "502.4", 2]
print(type(my_mixed_list)) # <class 'list'>
print(type(my_mixed_list[1])) # <class ‘str'>
```

LISTS

Similar to strings, there are list methods. You can find them here https://www.w3schools.com/python/python_ref_list.asp

List methods can only be performed on variables that have the data type list.

```python
fruits = ['apple', 'banana', 'cherry']
print(fruits) # ['apple', 'banana', 'cherry']
fruits.reverse()
print(fruits)# ['cherry', 'banana', 'apple']
```

LISTS METHODS

Append will add a new item to the list (to the end)

```
new_fruit = input("What is another fruit?")
fruits.append(new_fruit)
print(fruits)
```

LISTS: APPEND

pop removes an element from the list at a specific index

```
fruits.pop(1)
print(fruits)
```

LISTS: POP

Len works the same as it does with strings, returns the length of the list

```
print(len(my_list))
```

LISTS: LEN

```
Insert an element at the a location, takes two params, index
and item to add
new_fruit = input("What is another fruit?")
fruits.insert(2, new_fruit)
print(fruits)
```

LISTS: INSERT

For / in loop can be used to iterate through each item in a list or in a range.

```
for item in a_list:
    #do task
```

In this example, item is the name we are calling the individual item. a_list is a reference to an actual list.

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(len(fruit))
```

LOOPS: FOR IN

Fruit represents a single item in the list. It changes as we iterate through the list. Every loop, we're looking at the next item of fruit inside fruits.

```
fruits = ['apple', 'banana', ‘cherry']
```

Fruits references the specific list

```
for fruit in fruits:
    print(len(fruit))
```

Fruit can be named anything, but this is typical naming convention.

FOR / IN LOOPS

We can turn a string into a list if it has a delineator (something to separate the items of a list. Once it has been made into a list, you can now use list methods on it.

```
# Convert a string into a list
my_string_to_convert = “apples,oranges,bananas"
print(my_string_to_convert) # apples, oranges, bananas,
my_string_to_convert = my_string_to_convert.split(“,") #
['apples', 'oranges', 'bananas’]
print(my_string_to_convert)
```

MAKE A STRING INTO LIST

Delineators can be anything! “this|is|my|string” or even “this is my string” where the delineator is a space.

This can be useful when getting data from a larger file that ou need to clean up. For example, data from a study or collection!

STRING TO LIST

Range returns a value that can be used to move through a set number of iterations

Here range is 0 - 6 (non inclusive)

```
for x in range(6):
  print(x)
# 0 1 2 3 4 5
```

Here range is 3 - 6 (non inclusive)

```
for x in range(3, 6):
  print(x)
# 3 4 5
```

LOOPS: FOR IN RANGE

With three values we can set the increment per loop, here I'm setting it to 2. So it will count by 2 (non inclusively)

```
for x in range(0, 6, 2):
  print(x, end=“ ")
# 0 2 4
```

Note: you can use end=", " to indicate how the line ends instead of a new line. Ex; here we put a ", " instead of a line break. This Is useful when debugging loops!

```
for i in range(-1, 5, 2):
    print(i, end=", ") # prints: -1, 1, 3,
```

LOOPS: FOR IN RANGE

For: Loops through a range of items, or for each item in a set of items

```
my_string = "this is my string"

for ch in my_string:
    print(ch)
```

For loop syntax, FOR / IN a specific space

Ch is variable that references each element, as we go through the loop.

my_stringThe specific string we want to loop through.

In this example, ch represents each element in my my_string.

FOR LOOPS: A STRING

Break is a way to end a loop.

If you return in a function, your loop will also end (and return from the function)

Looping through strings is useful because you can perform a larger action on each element of a string, rathe that relying on string methods to manipulate the string.

```
my_string = "this is my string"
for ch in my_string:
    if ch == "i":
        print("letter is i, so break")
        break
    else:
        print("letter isn't i")
```

BREAK // RETURN

```python
def pigLatinify(word):
    first_vowel = 0
    for ch in word:
        if ch.lower() in 'aeiouy':
            return word[first_vowel:] + '-' + word[:first_vowel] + 'ay'
        first_vowel = first_vowel + 1
    return word + '-ay' #handle no vowel case

print(pigLatinify("Hello")) # ello-Hay
```

FOR LOOPS: A STRING

```
# Find a range in a string
my_string = "A string to iterate through, lets find some letters"
start_position = my_string.find("string")
end_position = my_string.find("find")
# r = range(start_position, end_position)
for i in range(start_position, end_position):
    print(my_string[i], end= "  “)
```

This example uses range and for loop in a string.

We are using range to find the indexes in a string between two words and listing each letter between them. We can also start at the end of string by adding the length of the index of the word.

```
start_position = my_string.find("string")+len("string")
```

LOOPS: FOR IN RANGE

LAB TIME

Coming up: Nested loops

NEXT CLASS: