Introduction to Python

Fondren Library
Research Data Services

Walk-Through 3: Reverse a List

```
original_list = ['a', 'b', 'c', 'd', 'e']
reversed_list = []

for item in reversed(original_list):
    reversed_list.append(item)

print("Reversed list:", reversed_list)
print(original_list)
#Original List will print if you don't remove the second print
function

#append() = list_name.append(item) | Add items to the end of the list (numbers, strings, lists)
```

Walk-Through 4: Filtering a List

```
# Given a list of strings, we are going to write a loop to create a new list
# with only strings that have more than 3 characters.
words = ['cat', 'window', 'computer', 'no', 'yes', 'coffee', 'clothing',
'three']
long_words = []

for word in words:
    if len(word) > 3:
        long_words.append(word)
print("Words with more than 3 characters:" , long_words)
```

Group Activity: To-Do List Application in Python

1. Displaying the Menu: Write a function that prints out options for the user, such as add an item, view the list and remove an item.

2. Adding Items: Allow users to add new tasks to the to-do list.

3. Viewing the List: Display the current to-do list with each item numbered.

4. Removing Items: Let users remove tasks from the list by specifying the item number.

5. Exiting the App: Gracefully exit the application when the user is done.

Walk-Through (Creating a To-Do List):

```
# Group Activity: To-Do List Application in Python
def display_menu():
    print("To Do List Application")
    print("1. Add Item")
    print("2. View List")
    print("3. Remove Item")
    print("4. Exit")
    print()

def add_item(to_do_list):
    item = input("Enter the item to add to your to-do
list: ")
    to_do_list.append(item)

def view_list(to_do_list):
    print("\nTo-Do List:")
    for index, item in enumerate(to_do_list, start=1):
        print(f"{index}.{item}")
        print()
```

```
def remove_item(to_do_list):
    item_number = int(input("Enter the item number to remove: "))
    if 0 < item_number <= len(to_do_list):
        to_do_list.pop(item_number - 1)
    else:
        print("Invalid item number!")

def main():
    to_do_list = []
    while True:
        display_menu()
        choice = input("Choose an option: ")
        if choice == "1":
            add_item(to_do_list)
        elif choice == "2":
            view_list(to_do_list)
        elif choice == "3":
            remove_item(to_do_list)
        elif choice == "4":
            print("Goodbye!")
            break
        else:
            print("Invalid option, please try again.")

# Run the Application
if __name__ == "__main__":
    main()
```

DICTIONARIES

- Dictionaries in Python allow you to connect different pieces of information together.
- The data stored in dictionaries can be looped through similar to what we did in lists.
- Dictionaries allow you to store two kinds of information that can then be matched up.

- Follow along with examples.

DICTIONARIES

(X,Y) | (x, y)

Y = dependent value

X = independent value

patient_1 = {'first_name': 'John',
'last_name': 'Kimble'}
print (patient_1|'first _name']) print
(patient_1['last_name])

- The dictionary patient_1 stores the patients first name and last name. The two print statements display the information stored.

```
patient_1 = {'first_name': 'John', 'last_name': 'Kimble'}
```

```
print(patient_1['first_name'])
print(patient_1['last_name'])
```

```
John
Kimble
```

Dictionaries

```
Address_book = {
“Alice”: { “address” : “123 Maple St”, “phone”: “555-1234”},
“Bob”: { “address” : “456 Oak St”, “phone”: “555-4321”}
}
```

DICTIONARIES

- A dictionary in Python is a collection of key-value pairs.
- Each key is connected to a value. You can use a key to access the value associated with that key.
- A key's value can be a number, list, string, or even another dictionary.

DICTIONARIES – FOLLOW ALONG: 5 MINS

- Python Dictionaries are wrapped in braces, {}. With their key-value pairs inside the braces.
- A key-value pair is a set of values associated with each other.
- Every key is connected to its value by a colon, and individual key-value pairs are separated by commas.

```
patient_1 = {'first_name': 'John', 'last_name': 'Kimble'}
```

ACCESSING VALUES: 5 MINS

- To get the value associated with a key: give the name of the dictionary and place the key inside a set of square brackets

```
patient_1 = {'first_name': 'John', 'last_name': 'Kimble'}
```

```
print(patient_1['first_name'])
print(patient_1['last_name'])
```

```
John
Kimble
```

- You can have an unlimited # of key-value pairs in a dictionary.

ACCESSING DICTIONARIES: 2-3 MINS

```
patient_name = patient_1['first_name']
print("hello there, " + patient_name)
```

```
hello there, John
```

NEW KEY-VALUE PAIRS – 3-5MINS

- Data is always changing – or dynamic. It makes sense that you are able to modify your dictionaries.
- To add a new key-value pair, you would give the name of the dictionary followed by the new key in square brackets along with the new value.

```
patient_1['MRN'] = '000000000'
patient_1['Last_exam'] = 'echo'
```

```
patient_1
```

```
{'first_name': 'John',
 'last_name': 'Kimble',
 'MRN': '000000000',
 'Last_exam': 'echo'}
```

STARTING WITH AN EMPTY DICTIONARY: 3-5 MINS

- To start filling an empty dictionary, first define a dictionary

```
patient_2 = {}
```

- Then, add each key-value pair on its own line.

```
patient_2['first_name'] = 'Jean'
patient_2['last_name'] = 'picard'
```

```
print(patient_2)
```

```
{'first_name': 'Jean', 'last_name': 'picard'}
```

MODIFYING VALUES

- To modify a value,, give the name of the dictionary with the key in [ ]'s and then the new value you want associated with that key.

```
patient_2['first_name'] = 'Jean-Luc'
```

```
print(patient_2)
```

```
{'first_name': 'Jean-Luc', 'last_name': 'picard'}
```

REMOVING KEY-VALUE PAIRS

- Data stored within the key-value pair can be easily removed.
- You can use the **del** statement to completely remove a key-value pair – this is permanent.

```
del patient_2['first_name']
```

```
print(patient_2)
```

```
{'last_name': 'picard'}
```

```
shopping_cart = {
   "apples" : {"price" : 0.5, "quantity" : 10},
   "bread" : {"price" : 2.0, "quantity" : 4},
   "cucumbers" : {"price" : 0.4, "quantity" : 5},
   "coffee" : {"price" : 23, "quantity" : 1},
   "mustard" : {"price" : 2.5, "quantity" : 2},
   "milk" : {"price" : 5, "quantity" : 3},
}
```

Making a Change:

```
shopping cart = {
   "apples" : {"price" : 0.5, "quantity" : 10},
   "bread" : {"price" : 2.0, "quantity" : 4},
   "cucumbers" : {"price" : 0.4, "quantity" : 5},
   "coffee" : {"price" : 23, "quantity" : 1},
   "mustard" : {"price" : 2.5, "quantity" : 2},
   "milk" : {"price" : 5, "quantity" : 3},
}
shopping_cart["cucumbers"]["quantity"] = 7
print(f"The updated quantity of cucumbers is:
{shopping_cart['cucumbers']['quantity']}")
```

Add an Item:

```
shopping_cart['eggs'] = {"price": 3.0,
"quantity" : 2}
print(f"Eggs have been added:
{shopping_cart['eggs']}")
```

Calculate the Total Cost:

```
total_cost = 0
for details in shopping_cart.values():
    total_cost += details["price"] *
details["quantity"]
print(f"The total cost of the shoppint
is: ${total_cost}")
```

LIST OF FUNCTIONS FOR LISTS/DICTIONARIES

1. insert() - insert items at a specified position in a list
   a. list_name.insert(index, item)'
2. remove() - to remove the first occurrence of a specified item on a list/dictionary.
   a. list_name.remove(item)
3. pop() - remove and then return item at a specified position
   a. list_name.pop([index])
4. sort() - sort through a list in ascending order by default (custom sort key).
   a. list_name.sort([key=None], [reverse=False])
5. reverse() - reverse the order of items on a list
   a. list_name.reverse()
6. index() - returns the index of first occurrences of a specific set of items
   a. list_name.index(item)
7. count() - returns the number of items for a specified set of occurrences.
   a. list_name.count(item)
8. extend() - add elements to the current list (using a list, tuple, dictionary, etc) current
   a. list_name.extend(iterable)
9. clear() - remove the elements/items from a current list or dictionary, leaving it empty
   a. list_name.clear()
10. copy() - returns a shallow copy of a list or dictionary
    a. list_name.copy()
11. append() - to make a change to your list, adding or removing or reversing whatever the item is to the end of your list or dictionary
    a. list_name.append(item)

Office Hours - 8PM to 9PM