DATA SOCIETY:

Intermediate python - Control flow structures - 2

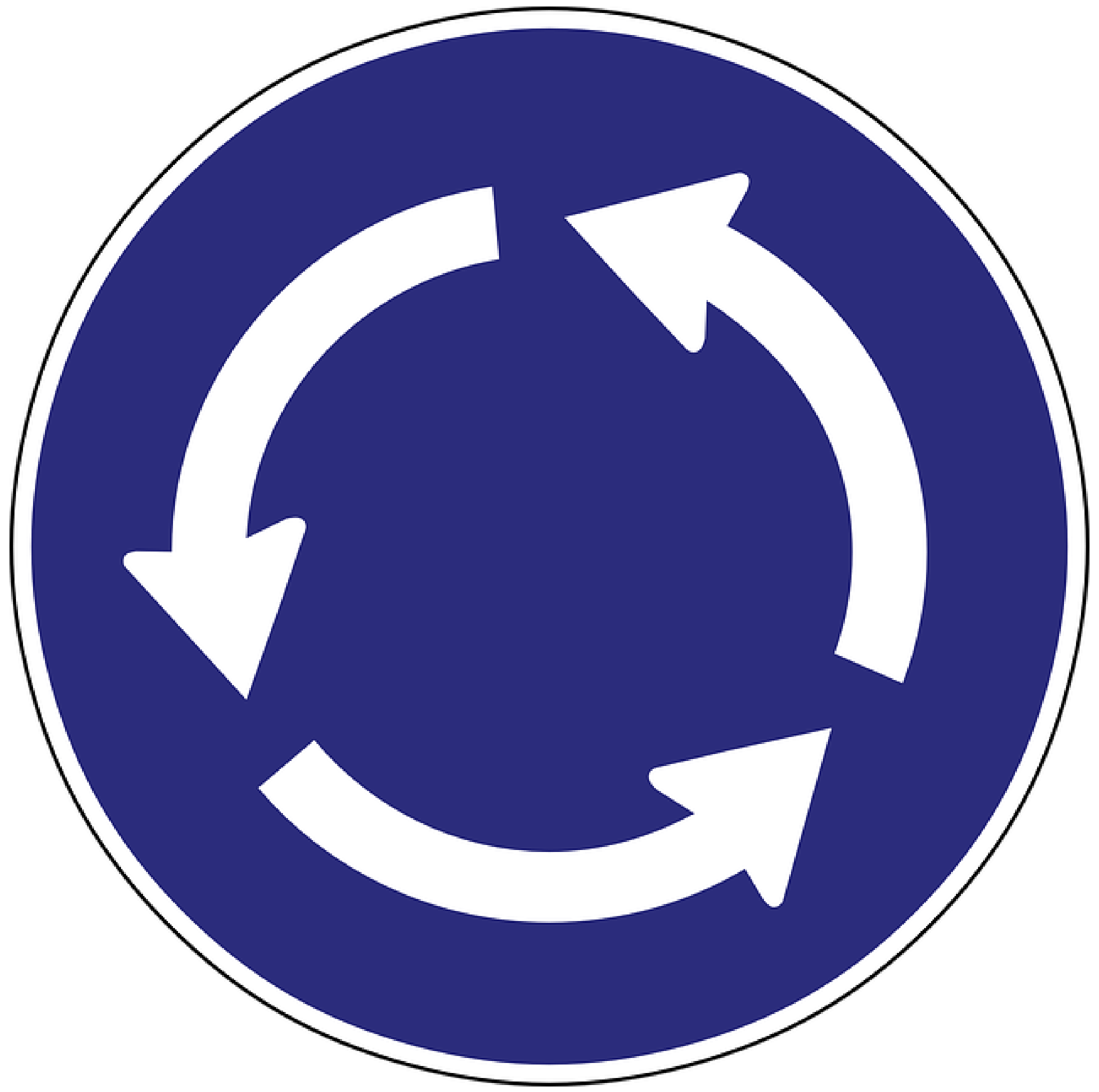
One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Implement for loops	
Implement list comprehensions	

Loops

- ***Loops*** allow our program to perform tasks over and over again given either:
  - A **counter** based on the number of actions we need to perform, or
  - A **condition**, based on which the loop will keep going until the condition no longer holds true
- In Python, we use `for` and `while` blocks to build *loops*
  - The loops using a **counter** are defined with `for` loops
  - The loops using a **condition** are defined with `while` loops



Loop scenario

- Imagine you have to organize a party
- You set up a calendar invite
- You need to send out invitations to each of your contacts
- For every contact in your contact list, you will need to perform the following actions:
    - copy the email address of the person
    - add it to a calendar invite
- **What kind of loop do you think you need for this?**

You need a for loop!

- **When should you use for loops?**
    - If you have a **finite** set of items you need to go through, with a definite start and end
    - If your items are organized in a *list*, *array*, *dictionary*, *sequence*, or any other **collection of elements**
- Our party planning programming need satisfies the conditions of a for loop:
    - We have a finite set of items to go through, i.e. the email addresses of each invitee
    - The items are organized as a list of contacts

For loops in Python

- To define a `for` loop in Python, we need 2 main components:

1. An **object** (e.g. a `list`) through which we would like to *iterate*
2. A **counter** variable that will increase based on the progress of the loop, or an **element** variable that takes on the value of next element in the collection of elements

Sample for loop - party invitations

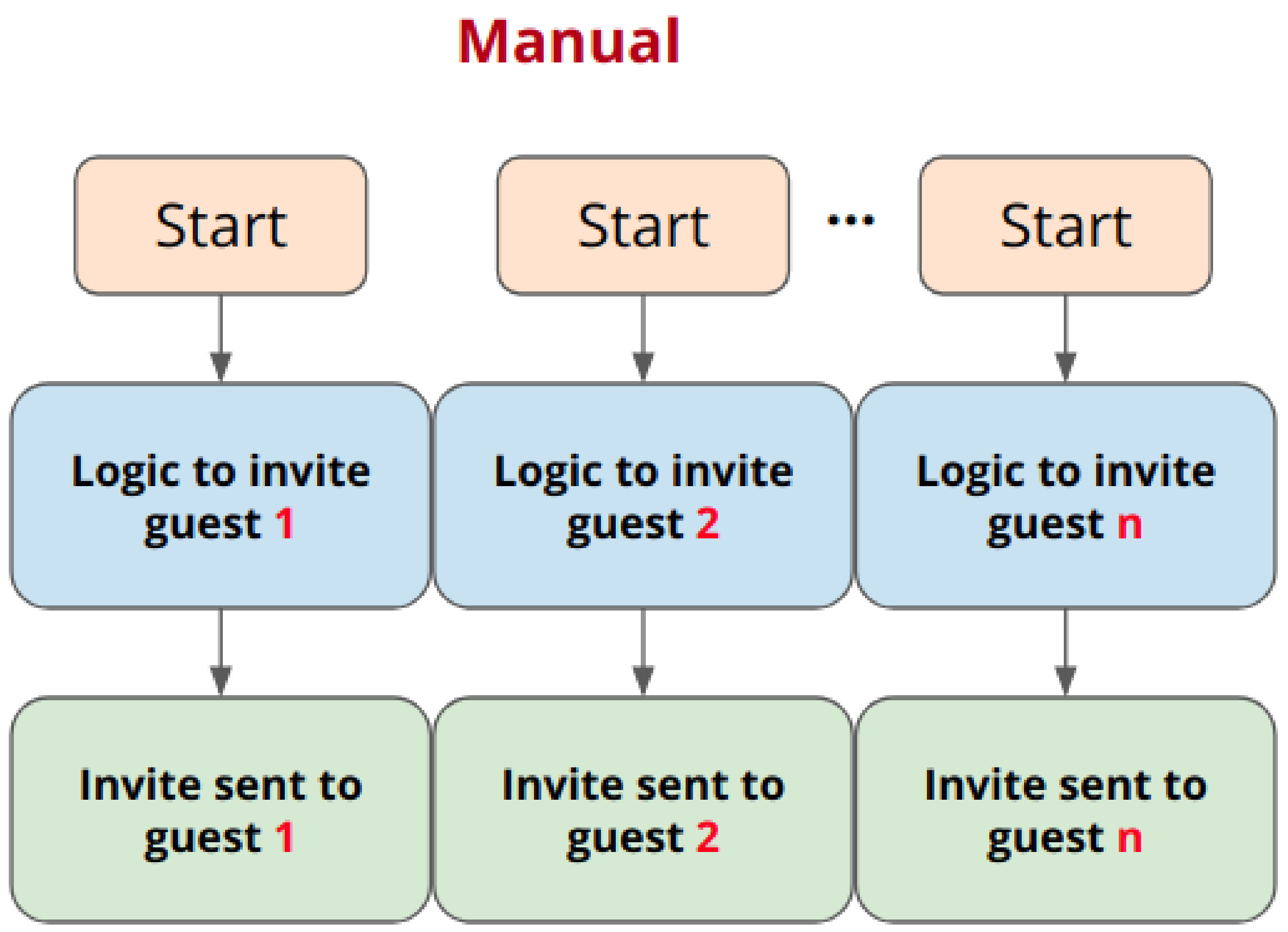
1. The **object** through which we *iterate* is a `contact_list`
2. The **element** of that object is a `name` variable that takes on a value from a `contact_list` at *each iteration* of the loop

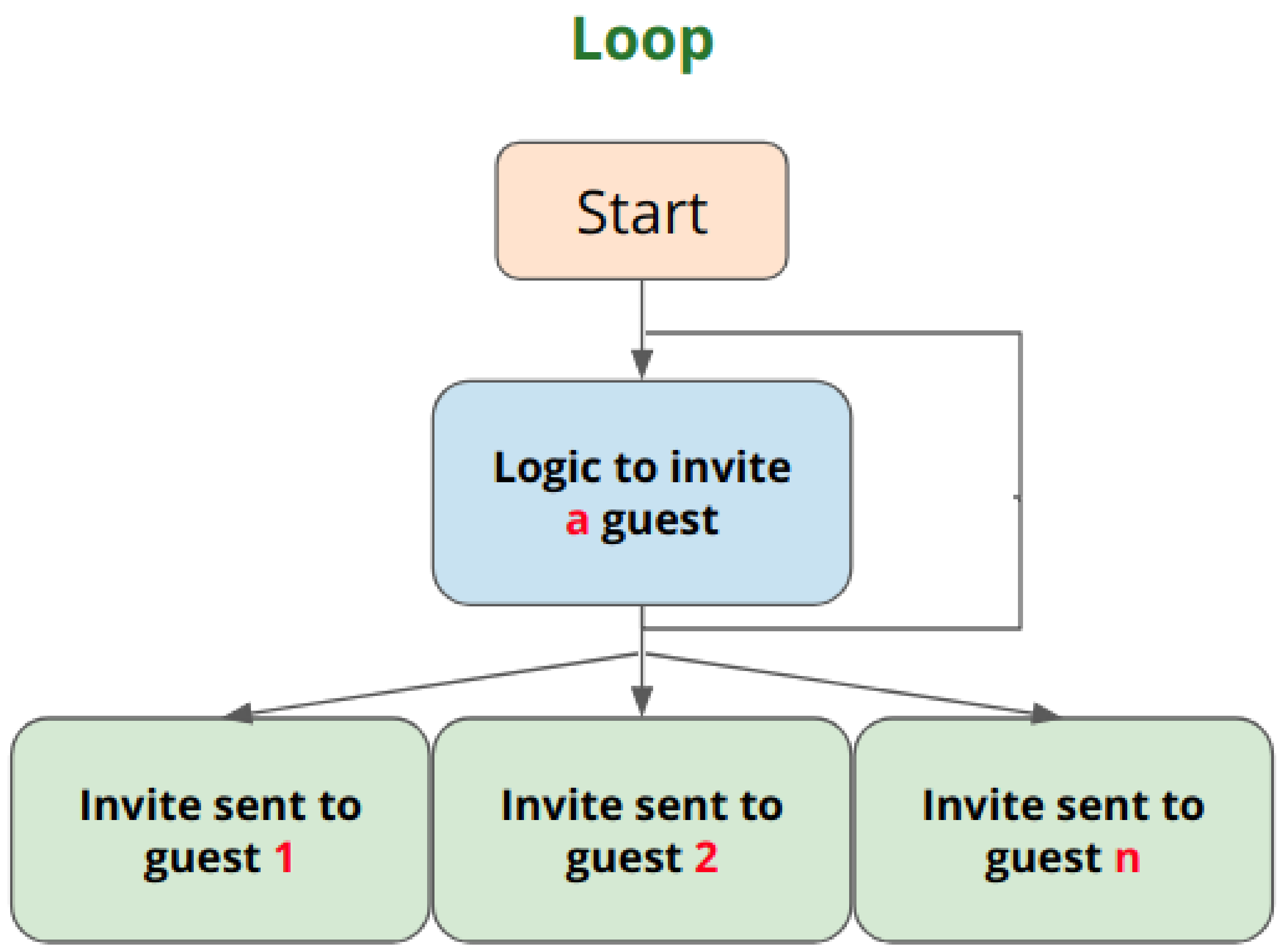
```
contact_list = ['Christian Bale', 'Bradley Cooper', 'Willem Dafoe', 'Rami Malek', 'Viggo
Mortensen', 'Yalitza Aparicio', 'Glenn Close', 'Olivia Colman', 'Lady Gaga', 'Melissa McCarthy']
```

```
for name in contact_list:
    print('Invite ' + name + '!')
```

```
Invite Christian Bale!
Invite Bradley Cooper!
Invite Willem Dafoe!
Invite Rami Malek!
Invite Viggo Mortensen!
Invite Yalitza Aparicio!
Invite Glenn Close!
Invite Olivia Colman!
Invite Lady Gaga!
Invite Melissa McCarthy!
```

For loop logic summary





Making a sequence of numbers in Python

- When working with loops, you will be looking at a lot of sequential elements and numbers
- To quickly create a sequence of numbers in Python, we will use the `range()` function
- When given a single number, `x`, this function will generate consecutive numbers from `0` to `x`, not including `x`
- We will also use `range()` to set the counter as an **index** to access the elements in the contact list

```
range(x)
```

Making a sequence of numbers in Python (cont'd)

- For example, to generate a sequence from `0` to `9` inclusive, you need to do the following:
    - `range(10)`

```python
sequence = range(10)
print(sequence)
```

```
range(0, 10)
```

```python
for number in sequence:
    print(number)
```

```
0
1
2
3
4
5
6
7
8
9
```

Making a sequence of numbers in Python (cont'd)

- To generate a sequence of numbers that starts with another number other than `0`, add a number as the first argument in `range()` method
  - `range(start, end + 1)`
- To generate a sequence of numbers from `1` to `10` inclusive
  - `range(1, 11)`

```python
sequence = range(1, 11)
print(sequence)
```

```
range(1, 11)
```

```python
for number in sequence:
    print(number)
```

```
1
2
3
4
5
6
7
8
9
10
```

Making a sequence of numbers in Python (cont'd)

- To generate a sequence of numbers with a step size different than `1`, you can add the third argument to `range()` function
  - `range(start, end + 1, step_size)`
- To generate a sequence of even numbers from `2` to `20` inclusive
  - `range(2, 21, 2)`

```
sequence = range(2, 21, 2)
print(sequence)
```

```
range(2, 21, 2)
```

```
for number in sequence:
    print(number)
```

```
2
4
6
8
10
12
14
16
18
20
```

For loops with counters in Python

- An alternative and a more general way to perform the same task of looping through a contact list is by using a **counter**
- In this case, we will use a counter as an **index** to access the elements of the `contact_list`

```
print(len(contact_list))
```

```
10
```

- To go through the entire list, we need to start at index `0` and end with index `9`, since there are a total `10` elements in the `contact_list`

```
# Save length of the list as number of contacts
# for convenience.
num_contacts = len(contact_list)
```

```
# Go through indices in a range between 0 and 9.
for i in range(num_contacts):
    # Invite a person in the list at index i.
    print('Invite ' + contact_list[i] + '!')
```

```
Invite Christian Bale!
Invite Bradley Cooper!
Invite Willem Dafoe!
Invite Rami Malek!
Invite Viggo Mortensen!
Invite Yalitza Aparicio!
Invite Glenn Close!
Invite Olivia Colman!
Invite Lady Gaga!
Invite Melissa McCarthy!
```

Using a for loop to build an object

- What if we need to **build an object** instead of running through an existing one? Like build a list with squares of numbers?
- We don't know what those elements should be, but we know 2 things
  - The **base for our squares**: numbers from 5 through 15
  - The **formula**: x^2

- We can create an empty list of `squares`, and a **range of numbers** from `5` through `15` (specified in the next slide)

```
squares = []
```

Using a for loop to build an object (cont'd)

- **Apply the formula** to each number `x`, getting an `x_squared` value
- Append the `x_squared` value to the list of `squares`

```
for x in range(5, 16):
    x_squared = x**2
    squares.append(x_squared)
    print("Square of", x, "is", x_squared)
```

```
Square of 5 is 25
Square of 6 is 36
Square of 7 is 49
Square of 8 is 64
Square of 9 is 81
Square of 10 is 100
Square of 11 is 121
Square of 12 is 144
Square of 13 is 169
Square of 14 is 196
Square of 15 is 225
```

```
print(squares)
```

```
[25, 36, 49, 64, 81, 100, 121, 144, 169, 196,
225]
```

Module completion checklist

Objective	Complete
Implement for loops	✔
Implement list comprehensions	

List comprehension

- Since `for` loops are tightly connected to finite collections of elements like **lists**, there exists a shorthand for creating lists using `for` loops called **list comprehension**
- List comprehension simplifies your code and makes it more concise
- It is more computationally efficient than the example on the previous slide

```python
squares = [x**2 for x in range(5, 16)]
print(squares)
```

```
[25, 36, 49, 64, 81, 100, 121, 144, 169, 196,
225]
```

Looping over a dictionary

- When you iterate through **dictionaries**, keep in mind that a dictionary is a list of *tuples*
- Take this dictionary of fare prices for different modes of transportation, for instance

```
prices = {'bus': 1.75, 'metro': 3.50, 'uber': 8.75, 'lyft': 7.50}
```

- We cannot use a simple index to access an element of the `prices` like this: `prices[i]`
- We will get a `KeyError` because we must access dictionary elements using a `key`
- Instead, we can use the `items()` method, which extracts the `keys` and `values` from each tuple

```
for key, value in prices.items():
    print('The price for', key, 'is', value)
```

```
The price for bus is 1.75
The price for metro is 3.5
The price for uber is 8.75
The price for lyft is 7.5
```

Knowledge check

Link: ***https://forms.gle/3DEdAsyUaPm1aikf8***

Module completion checklist

Objective	Complete
Implement for loops	✔
Implement list comprehensions	✔

Congratulations on completing this module!

You are now ready to try Tasks 5-8 in the Exercise for this topic