Data Types

Collections: Lists

A list a **mutable** collection of ordered items, that can be of mixed type - created using square brackets.

List examples

In [ ]:
```
# Define a list
lst = [1, 'a', True]
```

In [ ]:
```
# Print out the contents of a list
print(lst)
```

In [ ]:
```
# Check the type of a list
type(lst)
```

Indexing

Indexing refers to selecting an item from within a collection, and is done with square brackets.

In [ ]:
```
# Define a list
my_lst = ['Julian', 'Anne', 'Richard', 'George', 'Timmy']
```

In [ ]:
```
# Indexing: Count forward, starting at 0, with positive numbers
print(my_lst[2])
```

In [ ]:
```
# Indexing: Count backward, starting at -1, with negative numbers
print(my_lst[-1])
```

In [ ]:
```
# Indexing: Grab a group of adjacent items using `start:stop`, called a slice
print(my_lst[2:4])
```

Index Practices

```
In [ ]: # Define a list for the examples
        example_lst = [1, 2, 3, 4, 5]
```

```
In [ ]: example_lst[2]
```

```
In [ ]: example_lst[-3]
```

```
In [ ]: example_lst[1:3]
```

Clicker Question #1

What will be the output of the following piece of code:

In [ ]:
```
q1_lst = ['a', 'b', 'c', 'd']
q1_lst[-3:-1]
```

- a) 'a', 'b', 'c'
- b) 'c', 'b', 'a'
- c) 'c', 'b'
- d) 'b', 'c', 'd'
- e) 'b', 'c'

Clicker Question Answer

```
In [ ]: q1_lst = ['a', 'b', 'c', 'd']
        q1_lst[-3:-1]
```

- Negative indices index backwards through a collection
- A sequence of indices (called a slice) can be accessed using start:stop
    - In this contstruction, `start` is included then every element until `stop`, not including `stop` itself

SideNote: But why is it like this...

Starting at zero is a convention (some) languages use that comes from how variables are stored in memory , and 'pointers' to those locations.

Length of a collection

```
In [ ]:
# Define a new list
another_lst = ['Peter', 'Janet', 'Jack', 'Pam', 'Barbara', 'Colin', 'George']

# Get the length of the list, and print it out
print(len(another_lst))
```

The `in` Operator

The `in` operator asks whether an element is present inside a collection, and returns a boolean answer.

```
In [ ]: # Define a new list to work with
        lst_again = [True, 13, None, 'apples']
```

```
In [ ]: # Check if a particular element is present in the list
        True in lst_again
```

```
In [ ]: # The `in` operator can also be combined with the `not` operator
        '19' not in lst_again
```

Practice with `in`

```
In [ ]: # Define a list to practice with
        practice_lst = [1, True, 'alpha', 13, 'cogs18']
```

```
In [ ]: 13 in practice_lst
```

```
In [ ]: False in practice_lst
```

```
In [ ]: 'True' in practice_lst
```

```
In [ ]: 'cogs18' not in practice_lst
```

Clicker #2

After executing the following code, what will be the value of `output` ?

```
In [ ]: ex2_lst = [0, False, 'ten', None]

        bool_1 = False in ex2_lst
        bool_2 = 10 not in ex2_lst

        output = bool_1 and bool_2
```

- a) True
- b) False
- c) This code will fail
- d) I don't know

Clicker Question Answer

```
In [ ]: ex2_lst = [0, False, 'ten', None]

        bool_1 = False in ex2_lst
        bool_2 = 10 not in ex2_lst

        output = bool_1 and bool_2
```

- The `in` operator checks whether an element is present in a collection, and can be negated with `not`

Mutating a List

Lists are mutable, meaning after definition, you can update and change things about the list.

```
In [ ]: # Define a list
        updates = [1, 2, 3]
```

```
In [ ]: # Check the contents of the list
        print(updates)
```

```
In [ ]: # Redefine a particular element of the list
        updates[1] = 0
```

```
In [ ]: # Check the contents of the list
        print(updates)
```

Collections: Tuples

A tuple is an **immutable** collection of ordered items, that can be of mixed type - created using parentheses.

Tuple Examples

```
In [ ]: # Define a tuple
        tup = (2, 'b', False)
```

```
In [ ]: # Print out the contents of a tuple
        print(tup)
```

```
In [ ]: # Check the type of a tuple
        type(tup)
```

```
In [ ]: # Index into a tuple
        tup[0]
```

```
In [ ]: # Get the length of a tuple
        len(tup)
```

Tuples are Immuatable

```
In [ ]: # Tuples are immutable - meaning after they defined, you can't change them
        tup[2] = 1
```

Strings as Collections

Strings act like mutable, ordered collections of homogenous elements - specifically characters.

In [ ]:
```
# Define a string
my_str = 'TheFamousFive'
```

In [ ]:
```
# Index into a string
my_str[2]
```

In [ ]:
```
# Ask if an item is in a string
'Fam' in my_str
```

In [ ]:
```
# Check the length of a string
len(my_str)
```

SideNote: using counters

```
In [ ]: # Initialize a counter variable
        counter = 0
```

```
In [ ]: counter = counter + 1
        print(counter)
```

```
In [ ]: counter = counter + 1
        print(counter)
```

Pulling it Together: Collections, Membership & Conditionals

Clicker Question #3

What will be the value of `counter` after this code is run?

```
In [ ]: things_that_are_good = ['python', 'data', 'science', 'tacos']

        counter = 0

        if 'python' in thing_that_are_good:
            counter = counter + 1

        if len(thing_that_are_good) == 4:
            counter = counter + 1

        if things_that_are_good[2] == 'data':
            counter = counter + 1
```

```
a) 0   b) 1   c) 2   d) 3   e) 4
```

Clicker Question #4

What will be printed out from running this code?

```
In [ ]: lst = ['a', 'b', 'c']
        tup = ('b', 'c', 'd')

        if lst[-1] == tup[-1]:
            print('EndMatch')
        elif tup[1] in lst:
            print('Overlap')
        elif len(lst) == tup:
            print('Length')
        else:
            print('None')
```

a) EndMatch   b) Overlap   c) Length   d) Overlap & Match   e) None

Clicker Question Answer

What will be printed out from running this code?

```
In [ ]: lst = ['a', 'b', 'c']
        tup = ('b', 'c', 'd')

        if lst[-1] == tup[-1]:
            print('EndMatch')
        elif tup[1] in lst:
            print('Overlap')
        elif len(lst) == tup:
            print('Length')
        else:
            print('None')
```