# Lists and Tuples

Python Module 4

# **Compound Data Types**

- Basic (built-in) data types include: int, float, bool, str.
- Compound data types combine basic data types together.
- Built-in compound data types include: Lists (list),
   Tuples (tuple), Dictionaries (dict), and Sets (set).
- This lesson focuses on Lists and Tuples.

#### Lists

- A sequence of items separated by a comma, between []
- The items in a list can be of different types

```
mylist = [100, "hundred", "100", 2.14e3]
print(mylist)
```

[100, 'hundred', '100', 2.14e3]

#### List concatenation and repetition

#### Concatenating two lists

- Use the + operator
- Can be used to add items to an existing list

#### Repeating items

 Use the \* operator to repeat the list a given number of times

```
a = [1, "two", 3]
b = [["second", "list"], '7']
print(a + b)
[1, 'two', 3, ['second', 'list'], '7']
```

```
print([0] * 4)
```

```
[0, 0, 0, 0]
```

```
print([4, 2, 1] * 3)
```

```
[4, 2, 1, 4, 2, 1, 4, 2, 1]
```

# List indexing

- Different from Matlab!
- Each item can be referenced from start or end:

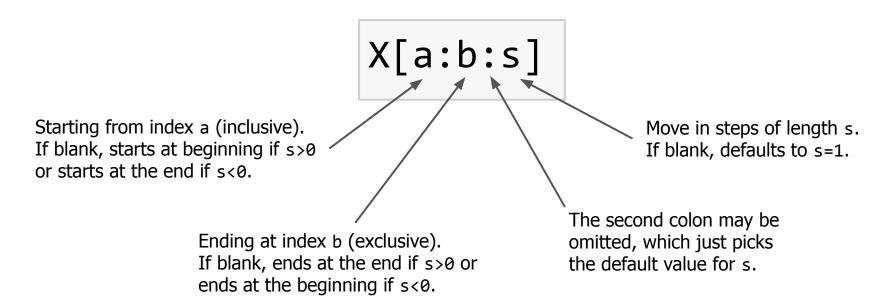
Starting from 0 counting from 
$$0$$
  $X = ['a', 'b', 'c', 'd', 'e', 'f', 'g']$   $0$   $1$   $2$   $3$   $4$   $5$   $6$  Starting from -1 beginning  $-7$   $-6$   $-5$   $-4$   $-3$   $-2$   $-1$  counting from the end

$$X[5] + X[-7] + X[-5] + X[4]$$

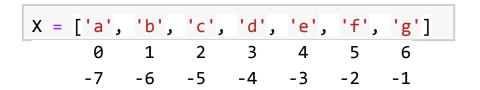
'face'

# List slicing

- Very different from Matlab!
- Can specify start, end, and step length.



### List slicing examples



X[1:4]

X[3:]

(click to reveal)

X[-5:5]

X[0:4:2]

X[:3]

X[3::-1]

#### Strings can be indexed and sliced too!

 Strings can be indexed and sliced just like lists

```
d = 'Data12'
print(d[2:5])
```

ta1

Lists are mutable:
 They can be changed

```
mylist = ['a', 'b', 'c', 'd', 'e', 'f']
mylist[3] = 'z'
print(mylist)
```

['a', 'b', 'c', 'z', 'e', 'f']

#### **WARNING:** mutable vs immutable

- For immutable objects: the variable points to the **object**
- For mutable objects: the variable points to the **container**
- To create a new object, make a copy as shown below

```
x = 5
y = x
x = 6
print(x,y)
```

```
6 5
```

```
X = ['a', 'b']
Y = X
X[0] = 'z'
print(X,Y)
```

```
['z', 'b'] ['z', 'b']
```

```
X = ['a', 'b']
Y = list(X)
X[0] = 'z'
print(X,Y)
```

```
['z', 'b'] ['a', 'b']
```

#### Common built-in functions for Lists

```
mylist = [-1, 34, 56, 2, -345]
# 1. Calculate the length of a list using the len() function.
len(mylist)
```

5

```
# 2. Get the minimum from a list using the min() function.
min(mylist)
```

-345

```
# 3. Get the maximum from a list using the max() function.
max(mylist)
```

56

#### Common built-in functions for Lists

```
# 4. Sum of the elements in the list using the sum() function.
sum(mylist)
```

-254

```
# 5. Sorting the list using the sorted() function. (this creates a COPY of the list) sorted(mylist)
```

[-345, -1, 2, 34, 56]

# **Tuples**

- Similar to a list, except tuples are immutable.
- Note the use of () parenthesis while defining tuples as compared to the [] used in lists.

```
DOB_record = ('Alicia', 'Smith', '3/12/1995')
print(DOB_record)

('Alicia', 'Smith', '3/12/1995')
```

Useful for indicating something that will not be changed.

# **Conversion between Tuples and Lists**

- Conversion between lists
   and tuples is possible using
   tuple() and list()
   built-in functions.
- These functions make a copy of the object.

```
mytuple = (1, 2, 3, 4, 5)
mylist = list(mytuple)
type(mylist)
```

list

```
mytuple_converted = tuple(mylist)
type(mytuple_converted)
```

tuple