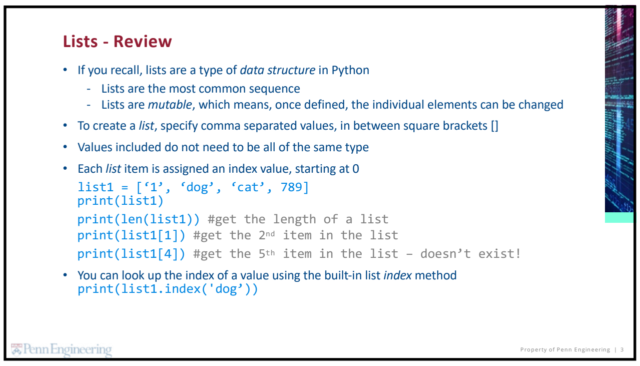
Lists & Strings

Brandon Krakowsky

1

Lists

2



Lists - Review

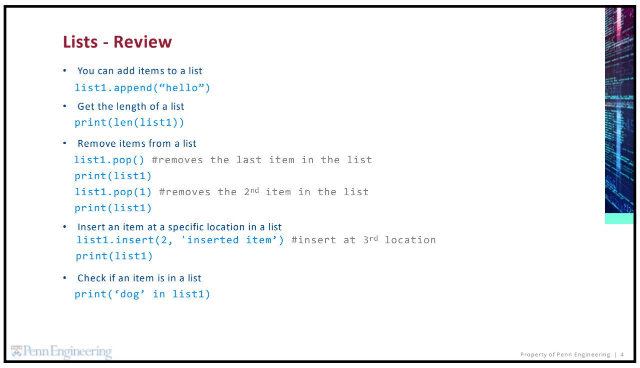
- If you recall, lists are a type of *data structure* in Python
  - Lists are the most common sequence
  - Lists are *mutable*, which means, once defined, the individual elements can be changed
- To create a *list*, specify comma separated values, in between square brackets []
- Values included do not need to be all of the same type
- Each *list* item is assigned an index value, starting at 0

```
list1 = [‘1’, ‘dog’, ‘cat’, 789]
print(list1)
print(len(list1)) #get the length of a list
print(list1[1]) #get the 2nd item in the list
print(list1[4]) #get the 5th item in the list – doesn’t exist!
```

- You can look up the index of a value using the built-in list *index* method

```
print(list1.index('dog’))
```

3



Lists - Review

- You can add items to a list

```
list1.append("hello")
```

- Get the length of a list

```
print(len(list1))
```

- Remove items from a list

```
list1.pop() #removes the last item in the list
print(list1)
list1.pop(1) #removes the 2nd item in the list
print(list1)
```

- Insert an item at a specific location in a list

```
list1.insert(2, 'inserted item') #insert at 3rd location
print(list1)
```

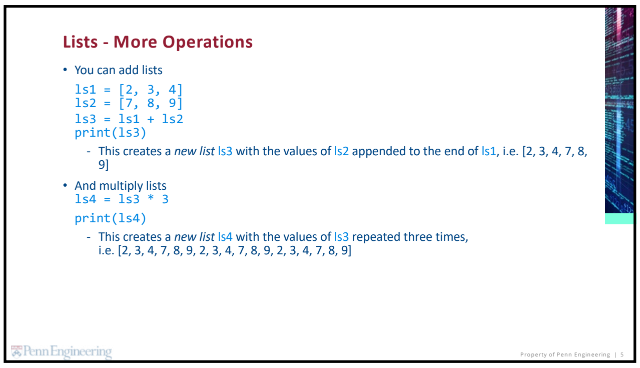
- Check if an item is in a list

```
print('dog' in list1)
```

Penn Engineering

Property of Penn Engineering | 4

4



Lists - More Operations

- You can add lists

```
ls1 = [2, 3, 4]
ls2 = [7, 8, 9]
ls3 = ls1 + ls2
print(ls3)
```

  - This creates a *new list* ls3 with the values of ls2 appended to the end of ls1, i.e. [2, 3, 4, 7, 8, 9]
- And multiply lists

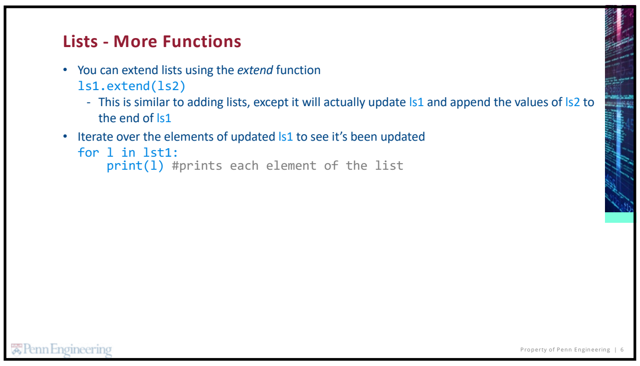
```
ls4 = ls3 * 3
print(ls4)
```

  - This creates a *new list* ls4 with the values of ls3 repeated three times, i.e. [2, 3, 4, 7, 8, 9, 2, 3, 4, 7, 8, 9, 2, 3, 4, 7, 8, 9]

Penn Engineering

Property of Penn Engineering | 5

5



Lists - More Functions

- You can extend lists using the *extend* function

```
ls1.extend(ls2)
```

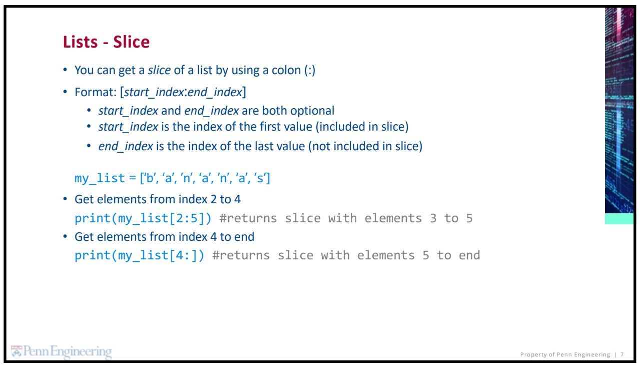
  - This is similar to adding lists, except it will actually update ls1 and append the values of ls2 to the end of ls1
- Iterate over the elements of updated ls1 to see it's been updated

```
for l in lst1:
    print(l) #prints each element of the list
```

Penn Engineering

Property of Penn Engineering | 6

6



Lists - Slice

- You can get a *slice* of a list by using a colon (:)
- Format: [*start_index*:*end_index*]
  - *start_index* and *end_index* are both optional
  - *start_index* is the index of the first value (included in slice)
  - *end_index* is the index of the last value (not included in slice)

```
my_list = ['b', 'a', 'n', 'a', 'n', 'a', 's']
```

- Get elements from index 2 to 4

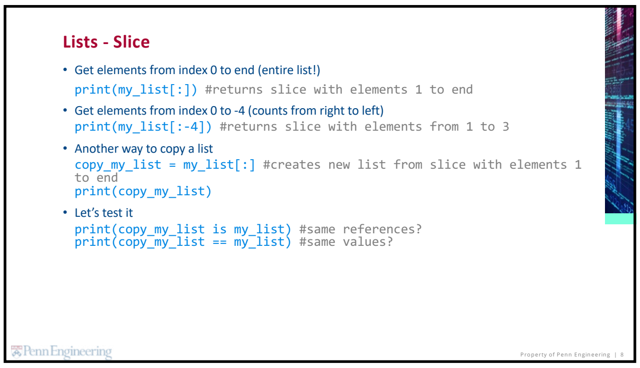
```
print(my_list[2:5]) #returns slice with elements 3 to 5
```

- Get elements from index 4 to end

```
print(my_list[4:]) #returns slice with elements 5 to end
```

Penn Engineering

7



Lists - Slice

- Get elements from index 0 to end (entire list!)

```
print(my_list[:]) #returns slice with elements 1 to end
```

- Get elements from index 0 to -4 (counts from right to left)

```
print(my_list[:-4]) #returns slice with elements from 1 to 3
```

- Another way to copy a list

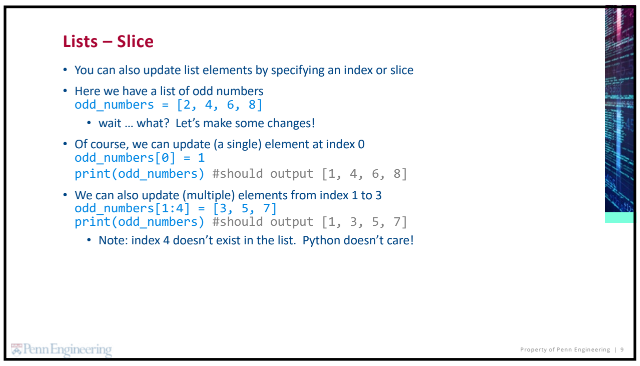
```
copy_my_list = my_list[:] #creates new list from slice with elements 1
to end
print(copy_my_list)
```

- Let's test it

```
print(copy_my_list is my_list) #same references?
print(copy_my_list == my_list) #same values?
```

Penn Engineering

8



Lists – Slice

- You can also update list elements by specifying an index or slice
- Here we have a list of odd numbers

```
odd_numbers = [2, 4, 6, 8]
```

  - wait … what? Let's make some changes!
- Of course, we can update (a single) element at index 0

```
odd_numbers[0] = 1
print(odd_numbers) #should output [1, 4, 6, 8]
```

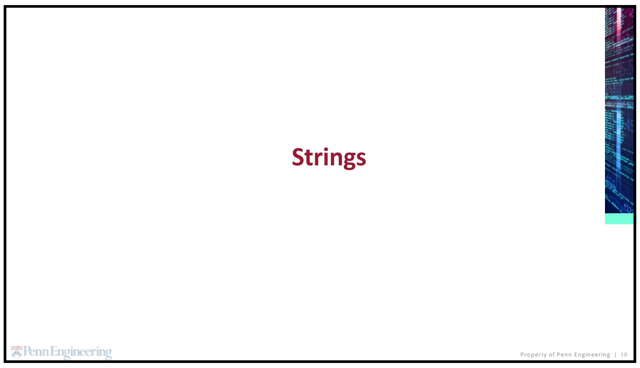
- We can also update (multiple) elements from index 1 to 3

```
odd_numbers[1:4] = [3, 5, 7]
print(odd_numbers) #should output [1, 3, 5, 7]
```

  - Note: index 4 doesn't exist in the list. Python doesn't care!

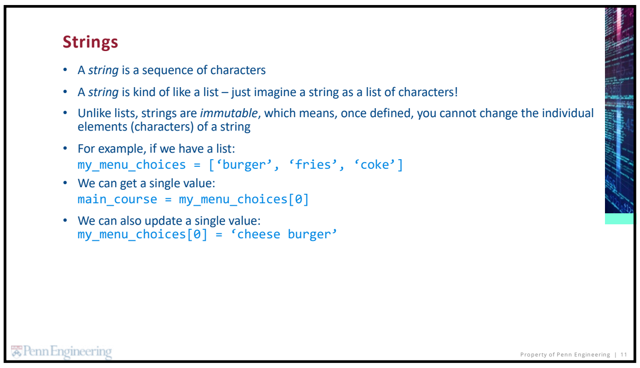
Penn Engineering

9



Strings

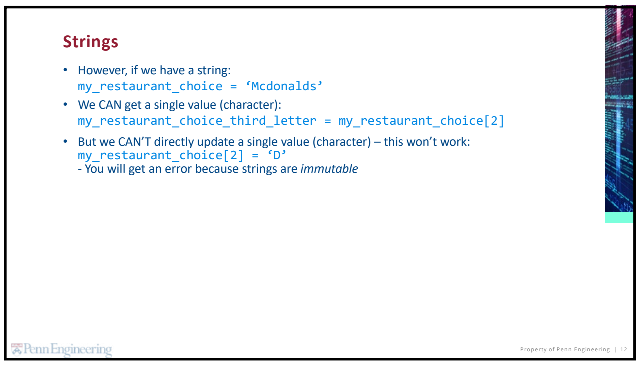
10



Strings

- A *string* is a sequence of characters
- A *string* is kind of like a list – just imagine a string as a list of characters!
- Unlike lists, strings are *immutable*, which means, once defined, you cannot change the individual elements (characters) of a string
- For example, if we have a list:
  `my_menu_choices = [‘burger’, ‘fries’, ‘coke’]`
- We can get a single value:
  `main_course = my_menu_choices[0]`
- We can also update a single value:
  `my_menu_choices[0] = ‘cheese burger’`

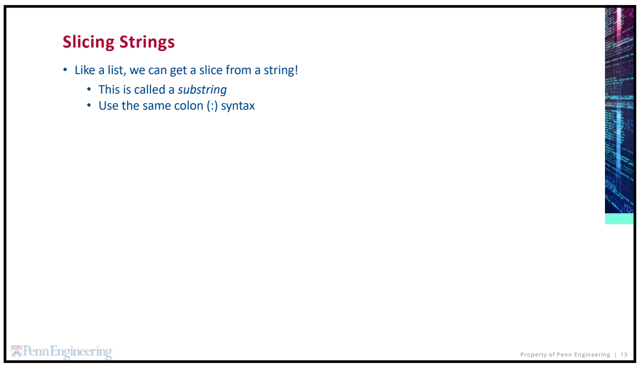
11



Strings

- However, if we have a string:
  `my_restaurant_choice = ‘Mcdonalds’`
- We CAN get a single value (character):
  `my_restaurant_choice_third_letter = my_restaurant_choice[2]`
- But we CAN’T directly update a single value (character) – this won’t work:
  `my_restaurant_choice[2] = ‘D’`
  - You will get an error because strings are *immutable*

12



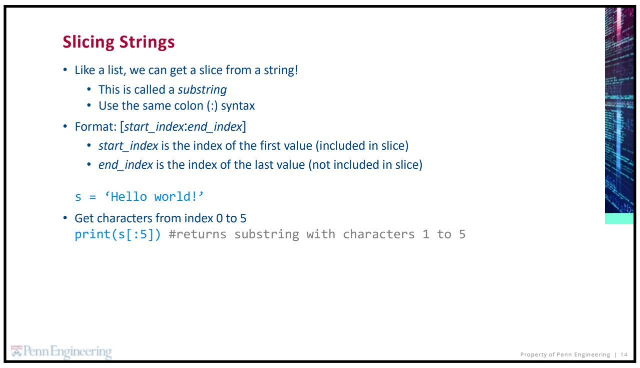
Slicing Strings

- Like a list, we can get a slice from a string!
  - This is called a *substring*
  - Use the same colon (:) syntax

Penn Engineering

Property of Penn Engineering | 13

13



Slicing Strings

- Like a list, we can get a slice from a string!
  - This is called a *substring*
  - Use the same colon (:) syntax
- Format: [*start_index*:*end_index*]
  - *start_index* is the index of the first value (included in slice)
  - *end_index* is the index of the last value (not included in slice)

```
s = ‘Hello world!’
```

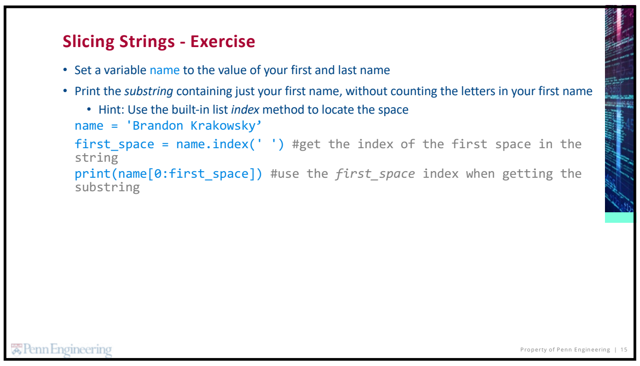
- Get characters from index 0 to 5

```
print(s[:5]) #returns substring with characters 1 to 5
```

Penn Engineering

Property of Penn Engineering | 14

14



Slicing Strings - Exercise

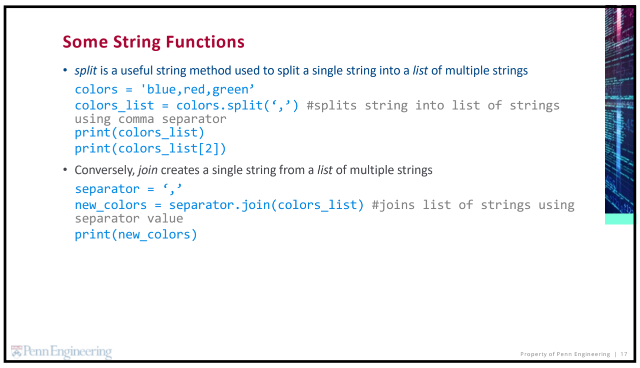
- Set a variable name to the value of your first and last name
- Print the *substring* containing just your first name, without counting the letters in your first name
  - Hint: Use the built-in list *index* method to locate the space

```
name = 'Brandon Krakowsky’
first_space = name.index(' ') #get the index of the first space in the
string
print(name[0:first_space]) #use the first_space index when getting the
substring
```

Penn Engineering

Property of Penn Engineering | 15

15



Some String Functions

- *split* is a useful string method used to split a single string into a *list* of multiple strings

```
colors = 'blue,red,green’
colors_list = colors.split(‘,’) #splits string into list of strings
using comma separator
print(colors_list)
print(colors_list[2])
```

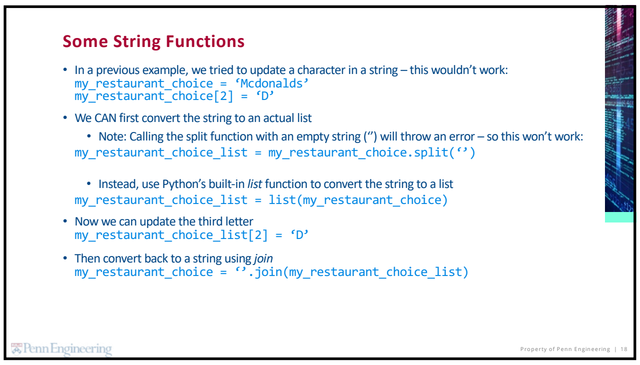
- Conversely, *join* creates a single string from a *list* of multiple strings

```
separator = ‘,’
new_colors = separator.join(colors_list) #joins list of strings using
separator value
print(new_colors)
```

Penn Engineering

Property of Penn Engineering | 17

16



Some String Functions

- In a previous example, we tried to update a character in a string – this wouldn’t work:

```
my_restaurant_choice = ‘Mcdonalds’
my_restaurant_choice[2] = ‘D’
```

- We CAN first convert the string to an actual list
  - Note: Calling the split function with an empty string (‘’) will throw an error – so this won’t work:

```
my_restaurant_choice_list = my_restaurant_choice.split(‘’)
```

  - Instead, use Python’s built-in *list* function to convert the string to a list

```
my_restaurant_choice_list = list(my_restaurant_choice)
```

- Now we can update the third letter

```
my_restaurant_choice_list[2] = ‘D’
```

- Then convert back to a string using *join*

```
my_restaurant_choice = ‘’.join(my_restaurant_choice_list)
```

Penn Engineering

Property of Penn Engineering | 18

17