Tuples & Mutability

CSE 160 Summer 2026

Questions?

Sli.do #cse160

Announcements

- **Written Check-In 5 due tonight at 11:59pm**
- **Programming Practice 4 due Sunday, July 26 at 11:59 PM**
- **Homework Assignment 4 due Monday, August 3 at 11:59 PM**
  - Longer and historically more challenging than past homework assignments
  - Two weeks to complete homework!
- Resubmission Cycle 2 now available, **due Wednesday July 29th at 11:59pm**
  - Homework 1 and Homework 2 are eligible
  - Fill out the Google Form!

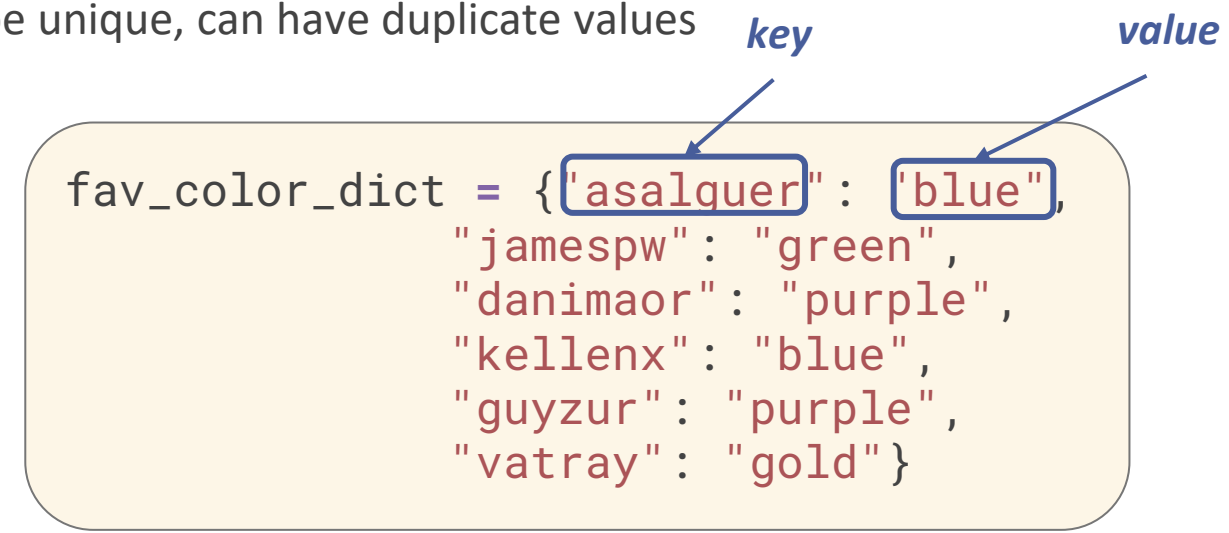
Today's Roadmap

1. Immutable Types
2. Tuples (an immutable type)
3. Assigning Variables vs. Mutating Objects

Jupyter Hub

Review: A Dictionary maps Keys to Values

- **Keys**:
  - Must be unique! No duplicate keys!
  - Must be an *immutable* type (e.g. integer, float, string)
- **Values**:
  - Do not need to be unique, can have duplicate values
  - Can be any type



Strings are an Immutable Type

But what does that mean?

Immutable: unchanging over time or unable to be changed

Q: Wait, can't I change or modify a string?

```
x = "hello"
x = "good bye"  # This just reassigns a different string
                # to the variable x. No string is modified.
```

A: **No** you cannot change or modify a string.

Q: o.k. maybe not that way, but didn't you tell us that many operations on lists also work on strings?

Many operations on Lists also work on Strings

Q: Remind me what operations can we do on lists?

Review: Querying Lists Summary

- What is the length of a list? `len(my_list)`
- Refer to a single element of a list `my_list[3]`
- Refer to a "slice" of a list `my_list[2:5]`
- Is a value in a list? `x in my_list`
- Where is a value in a list? `my_list.index(x)`
- How many of a value are in a list? `my_list.count(x)`

List operations that also work on Strings

```
a = "hello world"
print(a[0])
print(a[6])
print(len(a))
print(a[11])  # IndexError: string index out of range
print(a[-1])  # last character in string
print(a[1:3])
print(a[:])
print("w" in a)
print("z" in a)
print(a.index("d"))
print(a.index("z"))  # ValueError: substring not found
print(a.count("l"))
```

Python Tutor

Querying Strings Summary

- What is the length of `my_string`? `len(my_string)`
- Refer to a single character `my_string[3]`
- Refer to a "slice" of `my_string` `my_string[2:5]`
- Is string x in `my_string`? `x in my_string`
- Where is string x in `my_string`? `my_string.index(x)`
- How many of string x are in `my_string`? `my_string.count(x)`

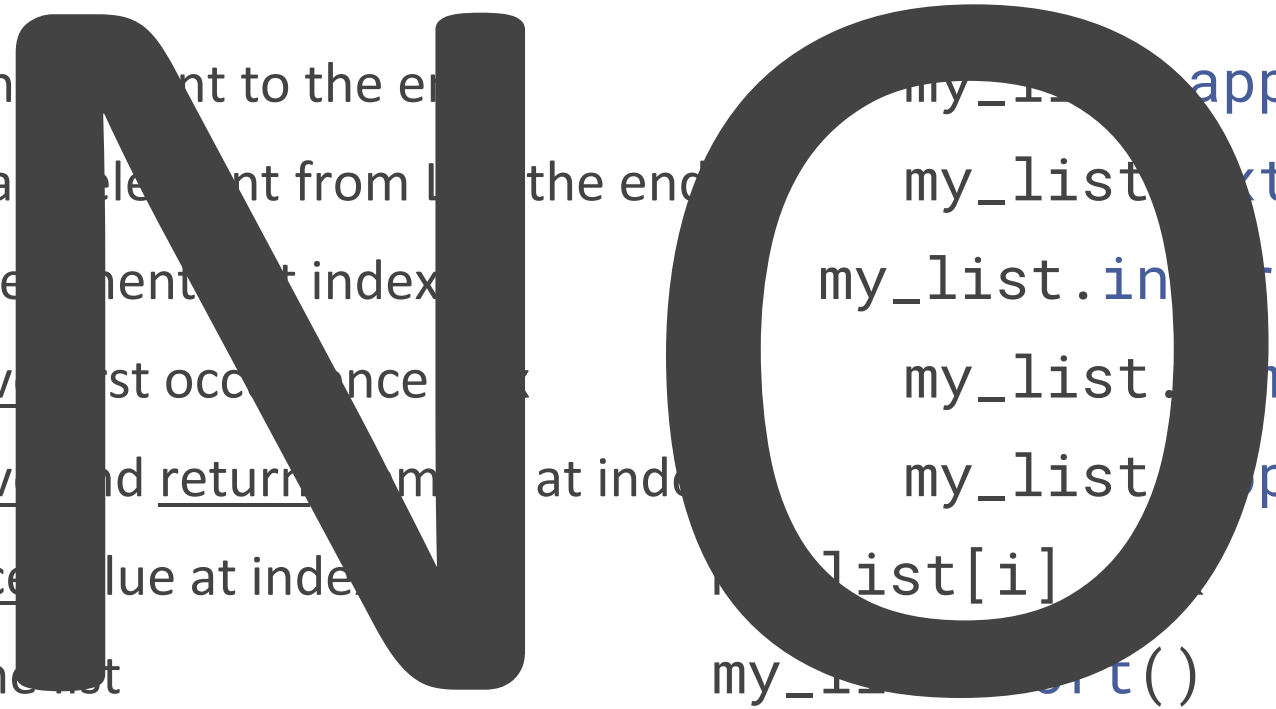
Review: Modifying Lists Summary

- Add an element to the end `my_list.append(x)`
- Add each element from L to the end `my_list.extend(L)`
- Insert element x at index i `my_list.insert(i, x)`
- Remove first occurrence of x `my_list.remove(x)`
- Remove and return element at index i `my_list.pop(i)`
- Replace value at index i `my_list[i] = x`
- Sort the list `my_list.sort()`
- Reverse the list `my_list.reverse()`

So can I modify a string using any of these?

- Add an element to the end `my_list.append(x)`
- Add each element from L to the end `my_list.extend(L)`
- Insert element x at index i `my_list.insert(i, x)`
- Remove first occurrence of x `my_list.remove(x)`
- Remove and return element at index i `my_list.pop(i)`
- Replace value at index i `my_list[i] = x`
- Sort the list `my_list.sort()`
- Reverse the list `my_list.reverse()`

So can I modify a string using any of these?



- Add an element to the end
- Add each element from L to the end
- Insert element at index
- Remove first occurrence of x
- Remove and return element at index
- Replace value at index
- Sort the list
- Reverse the list

```
my_list.append(x)
my_list.extend(L)
my_list.insert(i, x)
my_list.remove(x)
my_list.pop(i)
my_list[i] = x
my_list.sort()
my_list.reverse()
```

Strings are immutable

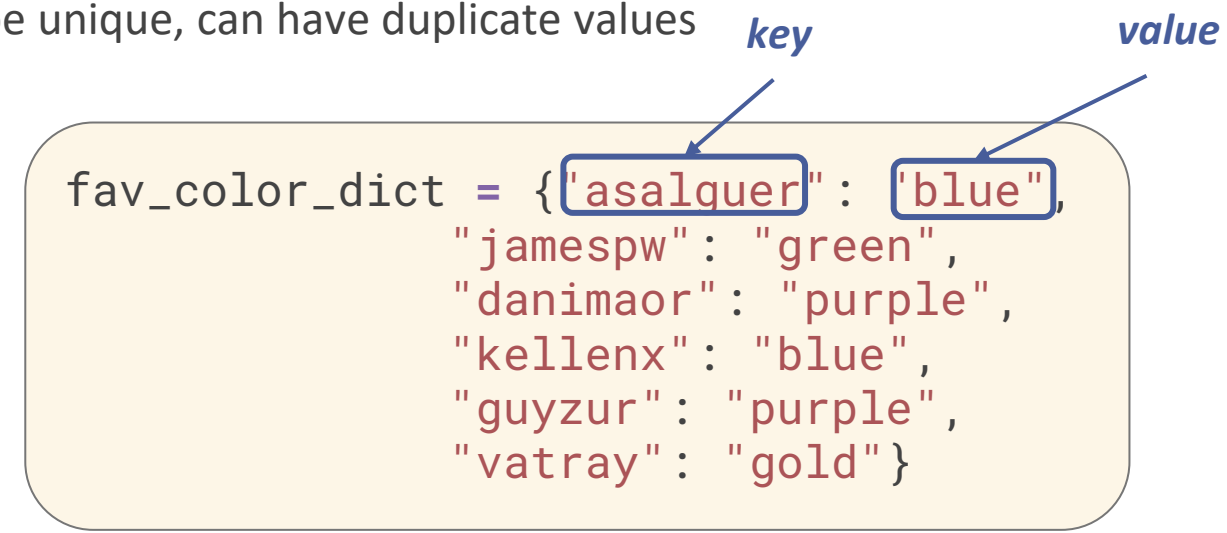
Q: Can you show me?

Python Tutor

Review: A Dictionary maps Keys to Values

- **Keys**:
  - Must be unique! No duplicate keys!
  - Must be an *immutable* type (e.g. integer, float, string, boolean, **tuple ***)
- **Values**:
  - Do not need to be unique, can have duplicate values
  - **Can be any type**

* Tuples containing only immutable types can be dictionary keys.

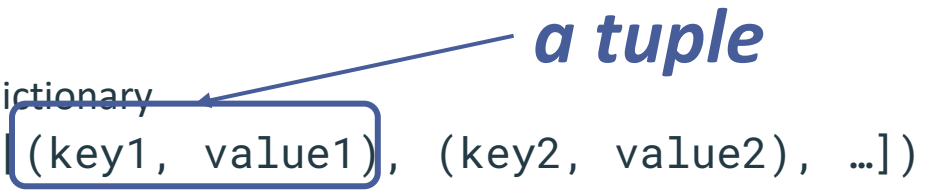


Q: What is a tuple?

A: You have actually already seen a tuple before…

Review: More Dictionary Methods

- **`dict_name.keys()`**
  - Retrieve all the keys in the dictionary
  - Returned in the form: `dict_keys([key1, key2,...])`
- **`dict_name.values()`**
  - Retrieve all the values in the dictionary
  - Return in the form: `dict_values([value1, value2,...])`
- **`dict_name.items()`**
  - Retrieve all the key-value pairs in the dictionary
  - Returned in the form: `dict_items([(key1, value1), (key2, value2), …])`



These are not lists, but a special dictionary object!
However, you can iterate through them using a for-loop just like a list!

Using `.items` on a dictionary to get tuples

```
fav_color = {“asalguer": "blue", "jamespw": "green",
             “danimaor": "purple", "kellenx": "blue",
             “guyzur": "purple", “vatray": "gold"}

for uwnetid, color in fav_color.items():
    print("favorite color of", uwnetid, "is:", color)
```

Python Tutor

A tuple is an immutable sequence

Python Tutor

- Lists, strings, and **tuples** are all ordered sequences
- Strings and **tuples** are *immutable*
- The elements of a **tuple** can be anything
  - (including mutable types)

```
# Creating tuples
w = (4, 7, 9)
x = ("once", "upon", "a", "time")
y = ()  # An empty tuple
z = “asalguer", "blue"  # parenthesis can make things
                        # clearer but are often not needed
```

Tuple Examples

Python Tutor

- Lists, strings, and **tuples** are all ordered sequences

```
w = (4, 7, 9)
print(w[0], w[1], w[2])
print(len(w))

x = ("once", "upon", "a", "time")
print(x[4])  # IndexError: tuple index out of range

y = ()  # An empty tuple
z = (19,)  # A tuple containing one item
num = (19)  # num is just an integer
```

Think Pair Share

Select **all** of the statements that will result in an error (or indicate that none do):

```
w = (4, 7, 9)
x = ("once", "upon", "a", "time")
```

A. `y = w[2]`

B. `print(x[3][2])`

C. `z = ([1, 2], [3, 4])`

D. `w[1] = 35`

E. None of these cause an error

Sli.do #cse160

Tuples are immutable

Q: Can you show me?

Python Tutor

Aside: Tuples in Homework 4

- The `read_data` function returns a tuple of lists:

```
# In utils.py
def read_data(fname):
    data = []
    label = []
    # (Code that modifies the two lists)
        # ...
    return data, label  # returns a tuple

# In analysis.py
data, label = read_data("data/mnist.csv")
```

Variable assignment vs. Object mutation

Assigning a **variable** changes what the variable is bound to**,** it does not change (mutate) an **object.**

```
size = 6  # Changes what the variable size is bound to
list2 = list1  # Changes what the variable list2 is bound to
```

Mutating (changing) an **object** does not change what any variable is bound to.

```
list1[2] = 5  # Modifies the object list1 refers to
list1.append(12)  # Modifies the object list1 refers to
```

Variable assignment vs. Object mutation Example

Python Tutor

```
list1 = [7, 8, 9]
list2 = ["e1", "e2"]

size = 6  # Changes/sets what the variable size is bound to
list3 = list2  # Changes what the variable list3 is bound to
list2 = list1  # Changes what the variable list2 is bound to

list1[2] = 5  # Modifies the object list1 refers to
list1.append(12)  # Modifies the object list1 refers to
```

Function Call Example

```
def change_val(lst):
    lst[0] = 13
def append_val(lst):
    lst.append(99)
def mystery(lst):
    lst = [78, 24]
    return lst

lst2 = [1, 2]
change_val(lst2)
append_val(lst2)
lst3 = mystery(lst2)
```

Python Tutor