CPSC 1101
Introduction to Computing

School of Engineering & Computing

Dept. of Computer Science & Engineering

Dr. Sidike Paheding

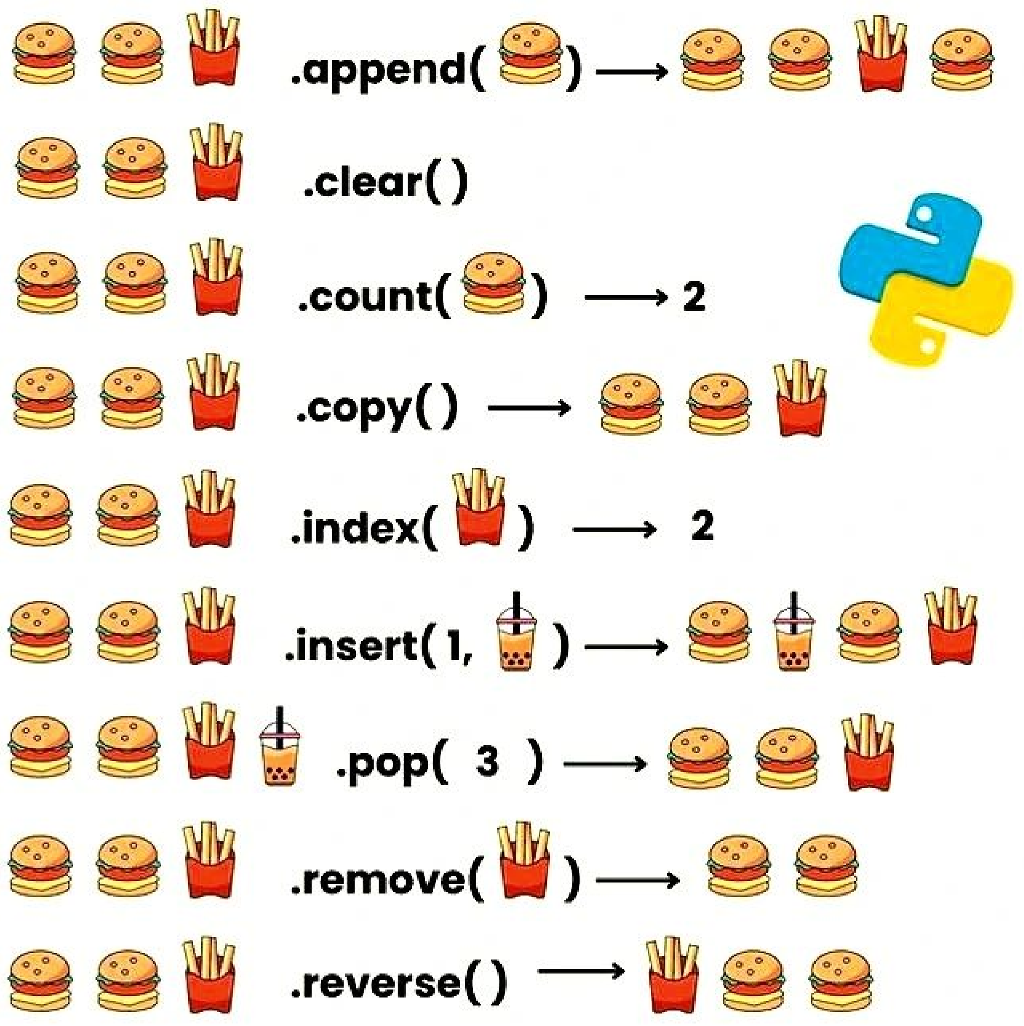
Agenda

- Recap Python List
- Lecture
- Homework – Read Chapter 6 and Chapter 12

Recap Python List

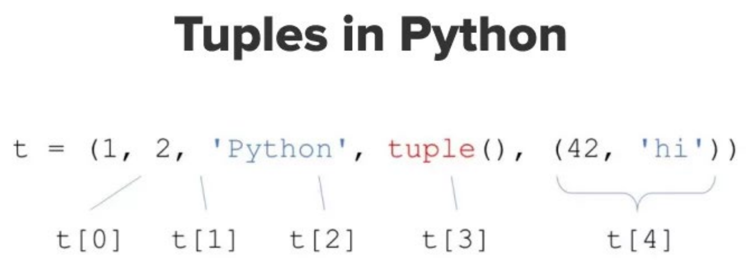
- A *list* contains a collection of *items. Use bracket [ ]*
- Use an *index* where **0** refers to the first item, **1** refers to the second item, and so on.
- -1 refers to the last item in the list, -2 refers to the second last item, and so on.
- Use the repetition operator (*) to repeat the items in a list.
- Use the methods append(), insert(), remove(), and pop() methods to modify lists.
- To process the items in a list, use *for/while* loops.

Recap Python List



Plickers Time!

Recap on the Python List



```python
t = (1, 2, 'Python', tuple(), (42, 'hi'))
```

Python Tuples

a collection of objects, which is ordered, immutable, indexed and allow duplicate values.

Python Tuples

- Tuples are ***ordered*** collections, which means the sequence of elements is defined and preserved.
- Tuples are also ***immutable***, indicating that once created, their contents <u>cannot</u> be modified, added to, or removed.
- Tuples are ***indexed***, allowing them to be accessed via an index.
- You can also store ***duplicate*** values at different positions.

Tuples

- A *tuple* is similar to a list:

```
tuple1 = (“apple”, ”orange”, ”mango”)
print(tuple1)
> ('apple', 'orange', 'mango’)
```

- They are immutable, like strings

```
tuple1[1] = “lemon”
TypeError: 'tuple' object does not support item assignment
```

Tuple Assignment

Values on right side

(a, b, c) = (4, 3, 6)

Tuple on left side

Same number of arguments!

a = 4

b = 3

c = 6

```
tuple_values = (1, 2, 3)
a, b, c = tuple_values          # a = 1, b = 2, c = 3
```

Assigning and Accessing Tuples

```
# Defining a tuple with mixed data types
my_tuple = (1, "Hello", 3.14)

# Accessing elements by index
print(my_tuple[0])  # Output: 1
print(my_tuple[1])  # Output: Hello
print(my_tuple[-1])  # Output: 3.14
```

Tuples Exercises

my_tuple = (1, 5, -2, 8, 9, 10)

```
# Accessing the elements
print(my_tuple[-2])
print(my_tuple[1:3])
print(my_tuple[-3:-1])
print(my_tuple[2:])
```

```
9
(5, -2)
(8, 9)
(-2, 8, 9, 10)
```

```
my_tuple[2] = 20
```

```
TypeError: 'tuple' object does not
support item assignment
```

Creating a tuple with a single item

```
Tuple1 = (1)
print(type(Tuple1))
print("\n Tuple is:",
Tuple1)
```

```
<class 'int'>
Tuple is: 1
```

Tuple 1 is not a tuple, as can be seen; instead, it is treated as an int, which is an issue.

```
Tuple1 = (1,)
```

The solution is to follow the item with a comma.

Tuple Methods: count() and index()

- numbers = (1, 2, 3, 2, 4, 2)

# count() method returns the number of times a specified element appears in the tuple.

- print(numbers.count(2)) # Output: 3

# index() method returns the index of the **first** occurrence of a specified element in the tuple.

- print(numbers.index(3)) # Output: 2

Reversing a Tuple

```
# Reversing elements in a tuple
fruits = ('apple', 'banana',
'cherry’)

print(fruits[::-1])
# Output: ('cherry', 'banana',
'apple')
```

Joining two tuples

```python
# Joining two tuples
fruits = ('apple', 'banana')
vegetables = ('carrot', 'potato')
combined = fruits + vegetables
print(combined)
```

```
('apple', 'banana', 'carrot', 'potato')
```

Tuple() method

The tuple() function is used to create a new tuple. It can be used in two ways:

- Without any argument: It creates an empty tuple.
- With an iterable argument: It converts the iterable (like a list, string, or dictionary) into a tuple.

```python
# Converting a list to a tuple
my_list = [1, 2, 3, 4]
my_tuple = tuple(my_list)

print(my_tuple)  # Output: (1, 2, 3, 4)
```

Tuple within a List

- a list can contain multiple tuples as its elements.

```
# List containing tuples
students = [
    ("Alice", 20, "A"),
    ("Bob", 22, "B"),
    ("Charlie", 21, "A"),
]

# Accessing elements
print(students[0])       # Output: ('Alice', 20, 'A')
print(students[1][0])    # Output: 'Bob'
```

List within a Tuple

- a tuple also can contain one or more lists as its elements.

```
# Tuple containing lists
shopping_list = (
    ["Milk", "Eggs", "Butter"],    # Dairy
    ["Apples", "Bananas", "Oranges"],  # Fruits
    ["Bread", "Rice", "Pasta"]    # Grains
)

# Accessing elements
print(shopping_list[0])        # Output: ['Milk',
'Eggs', 'Butter']
print(shopping_list[1][1])     # Output: 'Bananas'
```

Scenario: Restaurant Menu Management

- A restaurant’s menu is represented as a list of tuples. Each tuple contains information about a dish: its name, price, and availability.
  - How many times does the "Burger" appear on the menu?
  - What is the index of the first "Pizza" entry on the menu?
  - Find the total number of "In Stock" dishes on the menu.

```python
# Restaurant menu (list of tuples)
menu = [
    ("Pasta", 12.99, "In Stock"),
    ("Burger", 8.99, "Out of Stock"),
    ("Salad", 7.50, "In Stock"),
    ("Pizza", 11.99, "In Stock"),
    ("Burger", 8.99, "In Stock")
]
```

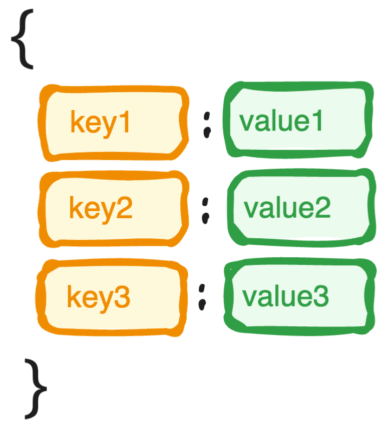
Scenario: Restaurant Menu Management

```python
# 1
# Extracting dish names from the menu into a tuple
dish_names = tuple([dish[0] for dish in menu])
print(dish_names)
# Use count() to find occurrences of 'Burger'
burger_count = dish_names.count("Burger")
print(burger_count)

# 2
pizza_index = dish_names.index("Pizza")

# 3
# Extracting availability status into a tuple
availability_status = tuple([dish[2] for dish in menu])
print(availability_status)

# Use count() to find the number of 'In Stock' dishes
in_stock_count = availability_status.count("In Stock")
print(in_stock_count)
```



Python Dictionaries

is a data structure that stores the value in **key: value** pairs.

A dictionary is a collection which is ordered*, changeable (mutable) and do not allow duplicates.

*As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

Dictionaries

A dictionary is a collection of key-value pairs.

```
d = {}
```

creates an empty dictionary, **d**

```
d[1996] = 'dog'
d[1995] = 'cat'
```

1996 is the key, **'dog'** is the value

1995 is the key, **'cat'** is the value

key value

A ***dictionary*** is a set of **key - value** pairs

```
print(d)
```

```
> {1996: 'dog', 1995: 'cat'}
```

```
print(d[1995])
```

```
> cat
```

As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

More on dictionaries

Strings can be keys, too!

```
d =  {1996: 'dog', 1995: 'cat'}
print('dog' in d) #False
print(1995 in d) #True
print(len(d)) #2
```

in checks if a key is present

len () returns the # of keys

```
print( d.keys())
> dict_keys([1996, 1995])
print(d.values())
> dict_values(['dog', 'cat'])
print(d.items())
> dict_items([(1996, 'dog'), (1995, 'cat')])
```

d.keys() returns a list of all keys

d.values () returns a list of all values

d.items () returns a list of all key, value ***pairs***

More on dictionaries (cont’d)

```
inventory = {'apples': 430, 'bananas': 312,
'oranges': 525, 'pears': 217}
del inventory['pears’]
print(inventory)
```

del removes a key-value pair

```
    {'apples': 430, 'bananas': 312, 'oranges': 525}
```

```
inventory['apples'] = 97
```

Dictionaries are mutable

```
  {'apples': 97, 'bananas': 312, 'oranges': 525}
```

Accessing Dictionary data

Syntax for accessing a value: *dictionary_name*[*key*]

```
countries = {
        'GB': 'United Kingdom (Great Britain)',
        'MX': 'Mexico’
}

#Code that gets a value from a dictionary
country = countries["MX"]# "Mexico"
country = countries["IL"]# KeyError: Key doesn't exist

#Code that sets a value if the key is in the dictionary
countries["GB"] = "United Kingdom"

#Code that adds a key/value pair if the key isn’t in the
dictionary
countries["FR"] = "France"
```

```
 >>  {'GB': 'United Kingdom (Great Britain)', 'MX': 'Mexico', 'FR': 'France'}
```

Check if value is in Dictionary data

Syntax for checking whether a key is in a dictionary: ***key* in *dictionary***

```
countries = {
       'GB': 'United Kingdom (Great Britain)',
       'MX': 'Mexico’
}
```

# Code that checks the key before getting its value

```
code = "IE"
if code in countries:
    country = countries[code]
    print(country)
else:
    print("There is no country for this code: “+ code)
```

Methods of Dictionaries

Method	**Parameters**	**Description**
keys	none	Returns a view of the keys in the dictionary
values	none	Returns a view of the values in the dictionary
items	none	Returns a view of the key-value pairs in the dictionary
get	key	Returns the value associated with key; None otherwise
get	key, alt	Returns the value associated with key; alt otherwise
copy	none	Returns a copy of the dictionary

Get() function of Dictionary object

`get(key[, default_value])`: If the specified **key** exists, this method returns its value. Otherwise, this method returns None or the default value if it is supplied.

```
countries = {
       'GB': 'United Kingdom (Great Britain)',
       'MX': 'Mexico'
}
```

```
#Code that uses the get() method          key
country = countries.get("MX") # "Mexico"
country = countries.get("NOT")# None
```

Delete function and Dictionary object

Syntax for deleting an item: `del dictionary_name[key]`

```
countries = {
        'GB': 'United Kingdom (Great Britain)',
        'MX': 'Mexico'
}
#Code that uses the del keyword to delete an item
del countries["MX"]
del countries["IEE"]        # KeyError: Key doesn't exist
#Code that checks a key before deleting the item
code = "IEE"
if code in countries:
    country = countries[code]
    del countries[code]
    print(country + " was deleted.")
else:
    print("There is no country for this code: " + code)
```

Additional delete functionality

`pop(key[, default_value])`: Returns the value of the specified key and deletes the key/value pair from the dictionary. The optional second argument is a value to return if the key doesn’t exist.

```
countries = {
        'GB': 'United Kingdom (Great Britain)',
        'MX': 'Mexico',
         'US': 'United States',
}
```

```
#Code that uses the pop() method to delete an item
```

```
country = countries.pop("US")   # "United States"
country = countries.pop("IEE")  # KeyError
country = countries.pop("IEE","Unknown")# "Unknown"
```

```
#Code that prevents a KeyError from occuring
```

```
code = "IEE"
country = countries.pop(code, "Unknown country")
print(country + " was deleted.")
```

```
Unknown country was deleted.
```

Additional delete functionality

```
clear(): Deletes all items.


countries = {
        'GB': 'United Kingdom (Great Britain)',

        'MX': 'Mexico',
        'US': 'United States',
}
#Code that uses the clear() method to delete all
items
countries.clear()   >> {}
```

Convert Dictionary to a List

```
countries = {"CA": "Canada", "US": "United
States", "MX": "Mexico"}
codes = list(countries.keys()) >> ['CA', 'MX', 'US']
codes.sort()
for code in codes:
    print(code + "     " + countries[code])
```

```
CA     Canada
MX     Mexico
US     United States
```

Convert List to a Dictionary

```
countries = [["GB", "United Kingdom"],
             ["NL", "Netherlands"],
             ["DE", "Germany"]]
countries = dict(countries)
print(countries)
```

The console

```
{'NL': 'Netherlands', 'GB': 'United Kingdom',
'DE': 'Germany'}
```

Unpack Tuple from a Dictionary

Python Dictionary items() Method

Return the dictionary's key-value pairs:

```
car = {
  "brand": "Ford",
  "model": "Mustang",
}
x = car.items()
print(x)
```

→

```
dict_items([('brand',
'Ford'), ('model',
'Mustang')])
```

```
for code, name in countries.items():
    print(code + "     " + name)
```

The console

```
MX      Mexico
US      United States
CA      Canada
```

Nested dictionaries

A dictionary can contain dictionaries, this is called nested dictionaries.

```
contacts = {
    "Joel":
        {"address": "1500 Anystreet", "city": "San Francisco",
         "state": "California", "postalCode": "94110",
         "phone": "555-555-1111"},
    "Anne":
        {"address": "1000 Somestreet", "city": "Fresno",
         "state": "California", "postalCode": "93704",
         "phone": "125-555-2222"},
    "Ben":
        {"address": "1400 Another Street", "city": "Fresno",
         "state": "California", "postalCode": "93704",
         "phone": "125-555-4444"}
}
```

Code that gets values from embedded dictionaries

```
phone = contacts["Anne"]["phone"]    # "125-555-2222"
email = contacts["Anne"]["email"]    # KeyError
```

Nested dictionaries: access items

```python
for name, info in contacts.items():
    print(f"{name}: {info['city']}")
```

```
Joel: San Francisco Anne:
Fresno Ben: Fresno
```

Nested dictionaries (cont.)

Code that checks whether a key exists within another key

```
key = "email"
if key in contacts["Anne"]:
    email = contacts["Anne"][key]
    print(email)
else:
    print("Sorry, there is no email address for this contact.")
```

Code that uses the get() method with nested dictionaries

```
phone = contacts.get("Anne").get("phone")      # "125-555-2222"
phone = contacts.get("Anne").get("email")      # None
phone = contacts.get("Mike").get("phone")      # AttributeError
phone = contacts.get("Mike", {}).get("phone") # None
```

If the key "Mike" does not exist in the contacts dictionary, the second argument {} (an empty dictionary) will be returned as the default value.

Exercise 1

Create a contact dictionary like the one below:

```
contacts = {
    "Joel":
        {"address": "1500 Anystreet", "city": "San Francisco",
         "state": "California", "postalCode": "94110",
         "phone": "555-555-1111"},
    "Anne":
        {"address": "1000 Somestreet", "city": "Fresno",
         "state": "California", "postalCode": "93704",
         "phone": "125-555-2222"},
    "Ben":
        {"address": "1400 Another Street", "city": "Fresno",
         "state": "California", "postalCode": "93704",
         "phone": "125-555-4444"}
}
```

Your contacts dictionary should contain 10 contacts.

1. **Print out all of the contact names**
2. **Locate all the contacts with the same postalCode "93704", and print their names and phone numbers**
3. **Replace the phone number "555-555-1111" with "111-555-1111", and print the updated contact information.**

Lists within Dictionaries

```
students = {"Joel":[85, 95, 70],
            "Anne":[95, 100, 100],
            "Mike":[77, 70, 80, 85]}
```

Code that gets a value from an embedded list

```
scores = students["Joel"]           # [85, 95, 70]
joel_score1 = students["Joel"][0]   # 85
```

Exercise 2

- You are given a dictionary where each key is a country code and the value is a list of major cities in that country. Your task is to iterate through this dictionary and print each country code along with its cities.

```
country_cities = {
    "US": ["New York", "Los Angeles", "Chicago"],
    "CA": ["Toronto", "Vancouver", "Montreal"],
    "GB": ["London", "Birmingham", "Manchester"],
    "FR": ["Paris", "Marseille", "Lyon"],
    "JP": ["Tokyo", "Osaka", "Kyoto"]
}
```

Fairfield UNIVERSITY

CPSC 1101 Introduction to Computing

School of Engineering & Computing

Dept. of Computer Science & Engineering

Dr. Sidike Paheding