INTRO TO PYTHON

SECTION 3

Emily Zhang
Senior Software Engineer

Lecture 2: Review

- If Statement
- Loops:
  - For loop
  - While Loop
- Function

Lecture 3: Outline

- Python Data Structures
  - Sequence
    - List
    - Tuple
  - Dictionaries
- Functions More

Sequence

Sequence Overview

- **Sequence**
  - **String**
  - **List**
  - **Tuple**
  - **Range**

- Other Sequence
  - byte sequences, byte arrays (won't cover)

Python List

- **Similar to 'array' in some languages**

- **List create**

```
>>> xs = [3, 1, 2]          # Create a list with square brackets, list index:  0, 1, 2
>>> xs                      # [3, 1, 2]
```

- **List index**

```
# list indices work similar to string indices, from 0 to length-1
>>> xs[-1]                  # Negative indices count from the end of the list; prints 2
>>> xs[2] = 'foo'           # Lists can contain elements of different types
>>> xs                      # [3, 1, 'foo']        # elements can have different types
```

List: Indexing & Slicing

```
# list indexing & slicing work similarly to string slicing

>>> nums = list(range(5))    # range is a built-in function that creates a list of integers
>>> nums                     # [0, 1, 2, 3, 4],  note, no '5'

>>> nums[2:4]                # Slice from index 2 to 4 (exclusive); prints [2, 3]
>>> nums[-2]                 # nums[-2] == nums[length-2], length=5, nums[-2] == nums[3]
>>> nums[:2]                 # Slice from the start to index 2 (exclusive); prints [0, 1]

>>> nums[:]                  # Slice of the whole list; prints [0, 1, 2, 3, 4]
>>> nums[:-1]                # Slice indices can be negative; prints [0, 1, 2, 3]
>>> nums[2:4] = [8, 9]       # Assign a new sublist to a slice, list is mutable
>>> nums                     # Prints [0, 1, 8, 9, 4]
```

List - Flexible Data Type

List	Description
a = [] or a = list()	An empty list
[1,1,2,3,5,8]	A list of integers
[42, "What's the question?", 3.1415, None]	A list of mixed data types
["Stuttgart", "Freiburg", "München"]	A list of Strings
all_tweets = [{ 'author': 'mary', 'handle': '@hadalittlelamb', 'date': '2013-01-22', 'tweets': ['at Loco Pops enjoying a Raspberry Sage popsicle', 'Learning Python is so much fun', 'Running late to Code and Coffee. #overslept',], },]	A mixed type, nested list List of dictionaries, which has key 'tweets' as lists

List Methods

```
# list methods
>>> xs = [3, 1, 2]
>>> xs.append('bar')          # Add a new element to the end of the list
>>> xs                        # Prints "[3, 1, 2, 'bar']"

>>> x = xs.pop()              # Remove and return the last element of the list
>>> x                         # Prints "bar "
>>> xs                        # prints [3, 1, 2]

# More methods
>>> dir(xs)
>>> [ x for x in dir(xs) if not x.startswith('__') ]      # str.startswith() method
['append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Object, Method

- **Object**
  - in python, everything's object, int, str, list, dict, tuple, set, ...
  - integer 1 is an object, **dir(1)** to list all attributes of this object
  - string 'hello' is an object, **dir('hello')** to list all attr

- **Method**
  - functions associated with objects, for example: **list.append(), str.format()**
  - access method by 'dot notation': **object.method()**, with "()" for function calling
  - sometimes, "chained dots": **module.object.method()**
  - **Very good tutorial:** https://www.programiz.com/python-programming/list

List Looping

```
>>> stocks = ['AAPL', 'FB', 'SNAP']          # list elements separated by “,”
>>> for s in stocks:                         # ‘for’ loop is the most common loop in Python
…       print('The next stock is: %s' % s)   # Prints each stock symbol on its own line.
…
The next stock is: AAPL
The next stock is: FB
The next stock is: SNAP
```

- **Each time, "s" gets a value from 'stocks'**
  - 1st loop, s='AAPL', 2nd loop, s='FB', ...
- **The entity after 'in' operator must be 'iterable'**
  - list, or something like a list that can return one value at a time (tuple, set, range, string)
  - it can not be a single value like a number or a string

Practice 1: list Loop

- We have a list of student scores, 'random' module to generate random numbers

```
>>> import random
>>> grades = [ random.randint(60, 100) for i in range(20) ]
```

- Write code to print only the score that's > 90

- Solution

```
import random
grades = [ random.randint(60, 100) for i in range(20) ]
for g in grades:
    if g > 90:
        print(g)
```

List Comprehension - Advanced

```
# list_comprehension.py

# Get new list by 'for loop'
nums = range(10)

new_list1 = []
for n in nums:
   if n%2==0:
      new_list1.append(n**2)


# Get new list by 'list comprehension'
new_list2 = [ x**2 for x in nums if x % 2 == 0 ]


# new_list1: [0, 4, 16, 36, 64]

# new_list2: [0, 4, 16, 36, 64]
```

```
# list comprehension:
# [ expression for item in a_list if something ]
numbers = range(10)

>>> new_list = [x for x in numbers]
>>> new_list
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> new_list = [x**2 for x in numbers if x>6]
>>> new_list
[49, 64, 81]

>>> new_list = [x for x in numbers if x%2!=0]
>>> new_list
[1, 3, 5, 7, 9]

>>> new_list = [str(x) for x in numbers if x>6]
>>> new_list
['7', '8', '9']
```

List & String

```
# string

>>> 'home' + 'work'     #  '+': concatenation
homework

>>> 'work' * 2          # *: repetition
workwork

>>> grades = 'ABCDE'
>>> grades[0]
'A'

>>> grades[2:4]          #  'CD'
>>> grades[:3]           #  'ABC'

>>> len(grades)
5
```

```
# list

>>> [1, 2] + [3, 4]      # "+" sign for concatenation
[1, 2, 3, 4]

>>> [1, 2] * 2           # "*" sign for repetition
[1, 2, 1, 2]

>>> grades = ['A', 'B', 'C', 'D', 'F']
>>> grades[0]
'A'

>>> grades[2:4]          #    ['C', 'D']
>>> grades[:3]           #    ['A', 'B', 'C']

>>> len(grades)
5
```

List ⟷ String

 # split: string → list

```
>>> s = "this is a string"
>>> l = s.split(" ")          # convert to a list of strings.
>>> l
['this', 'is', 'a', 'string']

>>> l2 = l = s.split()        # separates on spaces, \t, \n
>>> l2
['this', 'is', 'a', 'string']
```

 # join: list → string

```
>>> s2 = "-".join(l)
>>> s2
'This-is-a-string'

>>> s3 = "".join(l)
>>> s3 = 'thisisastring'

>>> s4 = " ".join(l)
'this is a string'
```

Practice 2: list vs. string

- We have a sentence: s = "this is a good day"

1. Convert the list to a list, called it: my_list

2. Convert my_list to string s1, separated by 1 space
3. Convert my_list to string s2, separated by 1 space
4. Convert my_list to string s3, separated by 1 space

- Solution for 1:
  - s = "this is a good day"
  - my_list = s.**split**()
  - print(my_list)

    my_list = ['this', 'is', 'a', 'good', 'day']

- Solution for 2/3/4:
  - s1 = " ".**join**(my_list)
  - s2 = " ".**join**(my_list)
  - s3 = "_".**join**(my_list)

Python Tuple

- Tuple is a **sequence**
  similar to list

- tuple is **immutable**
  can't change after create

- **create empty tuple**
  ```
  >>> a = tuple()    # or a = ()
  ```

- **tuple with 1 element**
  ```
  >>> b = (1,)     # note the comma
  >>> b
  (1,)
  ```

- **Indexing**
  ```
  >>> b[0]
  >>> b[-1]    # same as b[len(b)-1]
  ```

- **Update tuple: can not do direct update**

  ```
  >>>  a = (1, 2, 3)
  >>>  b = (4, 5, 6)
  >>>  c = a + b          # "+": concatenate, list/tuple/str
  >>>  c                  # c is a new tuple
  (1, 2, 3, 4, 5, 6)
  ```

- **Looping over Tuple**

  ```
  >>> my_sum = 0
  >>> for x in c:
  …       my_sum = my_sum + x   # accumulator pattern
  >>> my_sum
  21
  ```

Mutable vs. Immutable

```
# strings are immutable

>>> myString = "Hello World"
>>> myString[4]
'o'
>>> myString[2] = "p"

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

```
# tuples are immutable

>>> a = (1, 2, 3)
>>> type(a)
<class 'tuple'>
>>> a[1]
2
>>> a[1] = 4

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

```
# lists are mutable

>>> a = [1, 2, "3"]
>>> type(a)
<class list>
>>> a[1]
2
>>> a[1] = 4
>>> a
[1, 4, '3']
```

Sequence Summary

- **string, list, tuple**
- **common operation**: len(), looping, indexing, slicing
- List ←→ String: "split" and "join"
- list methods: append, pop
- tuple: similar to list, immutable, can't change after created
- mutable vs. immutable
- list comprehension (advanced)
  simple syntax and looks really neat, to create list from list

Dictionary

Dict Overview

- What is Dictionary
- Add / Remove Data
- Get Value by Key
- Looping over Dict
- Dict Methods
- Common Operations on Collections

What is Dictionary

- **Dict: a commonly used data type**
  - **List**: sequential, order is preserved
  - **Dict**, order is not-preserved, do not assume any order in dict
- **Index**
  - **List**: positions as index, [0: length-1]
    - list of **'values'**, ['john', 'mary', 'tom']
  - **Dict**: use a 'name' as index, we call this 'name' a 'key'
    - >>> dict_example = {**'name': 'john'**, **'age': 23**, **'height': 183**}
    - **key-value pairs**
- **Dict in other language**
  - hash, hash table, hash map, associative arrays, …
- **Dict: It's a natural way of organizing data**

List

index	value
0	"Eggs"
1	"Milk"
2	"Cheese"
3	"Yogurt"
4	"Butter"
5	"More Cheese"

Dictionary

key	value
'Eggs'	2.59
'Milk'	3.19
'Cheese'	4.80
'Yogurt'	1.35
'Butter'	2.59
'More Cheese'	6.19

- **List**: ['Eggs', 'Milk', 'Cheese', 'Yogurt', 'Butter', 'More Cheese', ……]
- **Dict**: {'Eggs': 2.56, 'Milk': 3.19, 'Cheese': 4.80, …… }

What is Dict Good For?

- **Storing very large data, dict offers fast lookup by key in O(1)**

  Using Dict: student grades, we can store grades in a dict, key-ed by name or id

```
grades = {
        "john": { "id": 1111111, "midterm": 98, "final": 96 },
        "kate": { "id": 2222222, "midterm": 86, "final": 77 }, …….
    }
```

  Using list:

```
grades = [(1111111, 'john', 98, 96), (2222222, 'kate', 86, 77).....]      # how to look up by name?
```

- **Some data in nature is mapping, not a list, and we don't care about order**
  - real dictionary: look up a word's definition by word
    i. the 'word' is the **key**, the 'definition' is the **value**
    ii. {'python': 'a high-level general-purpose programming language.'}
  - phone book: look up a what a phone number is for
    i. { '888-280-4331': 'amazon customer service',
       '866-540-3229': 'ebay customer service' }

Creating Dictionary

 # creating empty collections

- **Creating a List**

```
>>> a = list()
>>> b = [ ]
```

- **Creating a Tuple**

```
>>> c = tuple()
>>> d = ( )
```

- **Creating a Dict**

```
>>> e = dict()
>>> f = { }
```

 # creating collections with data

- **Creating a List**

```
>>> a = [1, 3, 5, 7, 9]
```

- **Creating a Tuple**

```
>>> c = ('a', 'b', 1, 3)
```

- **Creating a Dict**

```
>>> e = {'name': 'Sam'}
```

Rules About Dict

- **Colon to separate key/value, Comma to separate pairs**
  - {'Eggs': 2.56, 'Milk': 3.19, 'Cheese': 4.80}
  - {'name': 'john', 'age': 23, 'height': 183}

- **Allow Mixed Types**
  - {'apple': 'in stock', 1: 0, 'in stock': True}
  - {'name': 'john', 'scores': [99, 98, 97, 96, 95]}

- **Dict Keys Must Be Immutable Type**
  - allowed key types: int, string, tuple, …
  - not allowed: list, dict as dict keys

- **Dict Values Can Be Any Type**

Add Elements to Dict

- **Add Element to List, 'list.append' Method**

```
>>> my_list = [ ]
>>> my_list.append(5);  my_list                    #  [ 5 ]
```

- **Add Element to Dict, no such method, just add it directly**

```
>>> passwd = { }                                   # start empty dict
>>> passwd["bill"] = "bluescreen"
>>> passwd                                         #  {'bill': 'bluescreen'}
>>> passwd["bill"]                                 #  'bluescreen', get value by key using d[key]
```

Get Value by Key

- **Get value by key**

```
>>> d = {'name': 'john', 'age': 23, 'height': 183}
>>> d['name']
'john'
>>> d['age']
23
```

- Dict lookup is **super quick**
  - reason: **hashtable** implementation for python dict

- Dict is **unordered**, can **NOT** look up by position

```
>>> d = {'name': 'john', 'age': 23, 'height': 183}
>>> d[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 0
```

dict.get() method

- **Get value by key**

```
>>> d = {'name': 'john', 'age': 23, 'height': 183}
>>> d['name']
'john'
```

- **dict.get()** method returns a value for the given key. If key is not available then returns default value None.

```
>>> d.get('name')
'john'
>>> print(d.get('score', None))          # if key doesn't exist, return default
None
>>> print(d.get('score'))                # if no default, default is None
None
>>> d.get('score', 'UNKNOWN')            # default can be any str/number/……
UNKNOWN                                  # we often use: None, "", 0, [ ], { }, ….
```

Practice 3: Dict Get Value

- **Prepare the Dict**

```
>>> import random                          # import random module
>>> my_list = ['mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun']
>>> my_dict = { }
>>> for ea in my_list[:3]:
…         my_dict[ea] = random.randint(10, 20)   # random integer in [10, 20]
...
>>> my_dict                                # something like {'mon': 13, 'tue': 18, 'wed': 15}
```

- **Practice dict.get()**

1. for all keys in my_list, print corresponding values from my_dict, if missing, print None

   hint: use for loop over my_list

2. same as above, if missing, print empty string
3. same as above, if missing, print 'not found'
4. same as above, if missing, print boolean False

Practice 3: Solution

1. for all keys in my_list, print corresponding values from my_dict, is missing, return None

```
>>> for k in my_list:
…       print(my_dict.get(k, None))          # equivalent to print(my_dict.get(k))
```

2. same as above, if missing, return empty string

```
>>> for k in my_list:
…       print(my_dict.get(k, ""))
```

3. same as above, if missing, return 'not found'

```
>>> for k in my_list:
…       print(my_dict.get(k, "not found"))
```

4. same as above, if missing, return boolean False

```
>>> for k in my_list:
…       print(my_dict.get(k, False))
```

Dict: Mutable

- Similar to List, Dict is **Mutable**
  - mutable object types we learned: list, dict

- Given a key, you can assign a new value:

```
>>> passwd = { }
>>> passwd["bill"] = "bluescreen"
>>> passwd                                  #  {'bill': 'bluescreen'}
>>> passwd["bill"]                          #  'bluescreen'

# now assign a new value to the same key
>>> passwd["bill"] = "redsnake"
>>> passwd                                  #  {'bill': 'redsnake'}
>>> passwd["bill"]                          #  'redsnake'
```

Dictionary Methods

- **like other python objects, dict offers many methods**

- >>> dir(dict)
  ['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']

- blue color: what we will cover in this course

Delete Item from Dict

➔ **dict.pop(key)**

```
>>> d = {'name': 'john', 'age': 23, 'height': 183}
>>> d.pop('name')                          # 'pop' is a dict method, ( ) to call it
>>> d
{'age': 23, 'height': 183}
>>> d.pop('non_existing_key')              # 'pop' non existing key will raise KeyError
>>> d.pop('non_existing_key', None)        # 'pop' a key if exists, no error if not existing
```

dict.keys()

- **Prepare the dict**

```
>>> import random
>>> my_dict = { }
>>> for i in range(3):
…         my_dict[i] = round(random.random(), 2)      # float number in [0, 1)
…
>>> my_dict                                            # {0: 0.71, 1: 0.94, 2: 0.04}
```

- **dict.keys()**

```
>>> my_dict.keys()
dict_keys([0, 1, 2])
```

dict.values()

★ **Prepare the dict**

```
>>> import random
>>> my_dict = { }
>>> for i in range(3):
…         my_dict[i] = random.randint(10, 100)          # integers in [10, 100]
…
>>> my_dict                                              # {0: 16, 1: 62, 2: 63}
```

★ **dict.values()**

```
>>> my_dict.values()
dict_values([16, 62, 63])
```

★ **Do not rely on the orders in keys() and values() methods**

dict.items(), Practice 4

- **dict.items()**

```
>>> my_dict
{0: 0.71, 1: 0.94, 2: 0.04}
>>> my_dict.items()
dict_items([(0, 0.71), (1, 0.94), (2, 0.04)])    # (like) a list of tuples (key, value)
```

- **Practice 4**
  - print all key/value pairs in my_dict using **dict.keys()**

    >>> ???

  - print all key/value pairs in my_dict using **dict.items()**

    >>> ???

dict.items(), Practice 4 Solution

- **dict.items()**

```
>>> my_dict
{0: 0.71, 1: 0.94, 2: 0.04}
>>> my_dict.items()
dict_items([(0, 0.71), (1, 0.94), (2, 0.04)])          # returns tuple like of (key, value)
```

- **Practice**
  - print all key/value pairs in my_dict using **dict.keys()**

```
>>> for k in my_dict.keys():
...     print('key=%s, value=%s' % (k, my_dict[k]))   # get value by key: my_dict[k]
```

  - print all key/value pairs in my_dict using **dict.items()**

```
>>> for k, v in my_dict.items():                      # commonly used way to loop
...     print('key=%s, value=%s' % (k, v))
```

Looping Over Dict

- **Commonly Used Pattern**

```
>>> for k, v in my_dict.items():
...         pass
```

- **Loop by Keys**

```
>>> for k in my_dict.keys():
...         pass
```

- **Loop by Values**

```
>>> for k in my_dict.values():
...         pass
```

len() on collections

- **len():** builtin function

- **String Length**

```
>>> len( " Hello! How are you?! " )          # returns 20
```

- **List Length**

```
>>> len( [ x for x in range(20) ] )          # returns 20
```

- **Dict Length**

```
>>>  len( { "name": "jonny",  "role": "baba" } )          # returns 2
```

"In" operator

- **In Operator**
  - returns **boolean** type: True / False

- **string**
  - 'bc' in 'abcde' → True
  - 'in' operator for string checks if substring exists in a string

- **list**
  - 'a' in ['a', 'b', 'c']
  - 'in' operator in list: checks if an item exist in a list

- **dict**
  - >>> 'height' **in** {'name': 'john', 'age': 23} # returns False
  - check if a 'key' exist in a dict

Practice 5: User Password

- **Python Dict Usage Pattern**
  a. **Start with an empty collection**
     i. A common pattern for working with dict is to start with an empty collection
  b. **Add values to it one by one**
     i. using loops

- **Practice: username/password**
  a. Given a text file containing username and password one per line, Can we read it in and add them to a dictionary

```
$ cat dict_user_password.txt
john password1
mary password2
jose password3
```

Practice 5: hints

- Create a file under same directory of where you start Python: dict_user_password.txt, "user, pwd" pair separated by space

  ```
  john password1
  mary password2
  jose password3
  ```

- **open file:** dict_user_password.txt

  ```
  all_text = open('dict_user_password.txt').read()
  lines = all_text.split('\n')
  ```

- **start an empty dict, call it:** passwd

- **for each line, separate into 'user password' pair, split by space**
  - line example: john password1, how to separate string to list?

- **add each pair of data to dict:** passwd

Practice 5: Solution

- $ cat dict_user_password.txt

```
john password1
mary password2
jose password3
```

- Solution: dict_user_password.py

```
all_text = open('dict_user_password.txt').read()
lines = all_text.split('\n')
lines = [ x for x in lines if x ]      # remove empty str, "if x" means: if not empty

passwd = {}
for l in lines:
   user, pwd = l.split()
   passwd[user] = pwd                  # add key/value pair to a dict

print(passwd)
```

Practice 6: dict operation

- check type of your dict variable 'passwd'
  >>> ???
- get length of entries in a dict
  >>> ???
- get all the keys in a dict
  >>> ???
- get all the values in a dict
  >>> ???
- add an entry to a dict
  >>> ???
- remove entry 'john' from the dict
  >>> ???

Practice 6: Solution

- **check type of your dict variable 'passwd'**

```
>>> print(type(passwd))                    # note the function chaining
```

- **get length of entries in a dict**

```
>>> len(passwd)
```

- **get all the keys in a dict**

```
>>> passwd.keys()
```

- **get all the values in a dict**

```
>>> passwd.values()
```

- **add an entry to a dict**

```
>>> passwd['sofia'] = 'password4'
>>> passwd
```

- **remove entry 'john' from the dict**

```
>>> passwd.pop('john')
>>> passwd
```

Review: Dict Operations

- **dict.keys()**
  - returns a **list like** structure of all the **keys** of a dict
- **dict.values()**
  - returns a **list like** structure of all **values** of a dict
- **dict.items()**
  - returns a **list like** structure of all the **(key, value)** tuples
- **d1.update(d2) - advanced**
  - add all k/v pairs in d2 to d1
- **looping over dict**
  - **for k, v in dict.items():**

```
    pass                    #  do some operations of k, v
```

Practice 7: word count

- **Task: Count the frequency for each word in a text document?**
  - **Print something like:**

    The frequency for word 'the': 35

    The frequency for word 'a': 20

    …….

- **Prepare the text first**

```
>>> import this
... (zen of python) …
>>> text = """ ... (copy paste above text) … """          # triple quotes for multi line
>>> print(text)
```

Practice 7: Hints

- **What data type is in the text?**
  - **string**
    i. separated by spaces, punctuation signs, and newline "\n"
    ii. "this is one sentence.\nAnd this is the 2nd sentence"

- **How to generate a list from string?**
  - **string → list: str.split()**
  - **Example:**

```
>>> lines = text.split('\n')          # now we have list of lines, each line is a string
>>> for line in lines:
        words = line.split()          # words are list of words, may have mixed case and signs
        for w in words:
            w = w.lower()             # convert to all lower case
            w = w.strip(",-*.!")      # remove preceding and trailing sign from char list
            (add w to a list)         # how to add 1 value to a list?
```

Practice 7: Solution

full version see: **dict_word_count.py**

```
text = open('zen_of_python.txt').read()

all_words = []
lines = text.split('\n')
for l in lines:
   words = l.split()
   for w in words:
       w = w.lower()
       w = w.strip(",-*.!")
       all_words.append(w)

my_dict = {}
for w in all_words:
   my_dict[w] = my_dict.get(w, 0) + 1

for k, v in my_dict.items():
   print("the frequence of word: %s is: %s" % (k, v))
```

Zen of Python

- **Zen of Python**

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
……
```

Dictionary Summary

- dict is commonly used
- hash / hashtable in other languages
- create a dict
- add / remove entries
- keys / values / items
- length of dict
- loops for dict
- in operator

Dict Tutorials

Please pick one tutorial and follow it, make sure to understand what we learned

- https://www.python-course.eu/dictionaries.php
- https://www.datacamp.com/community/tutorials/python-dictionary-tutorial

Function

Function Can Call Another Function

```
# function_call_function.py

ORANGE_COUNTY_TAX_RATE = 0.08
TIP_FOR_GOOD_SERVICE = 0.2

def get_tip(bill_amount):
  return bill_amount * 0.20

def get_tax(subtotal):
  return subtotal * 0.0675

def print_receipt(subtotal):
  tax = get_tax(subtotal)
  bill_amount = subtotal + tax
  tip = get_tip(bill_amount)
  total = bill_amount + tip
```

Arguments with default values

```
# Also see: function_parameter_default_value.py
```

- **function argument can have default value(s)**
- **when calling a func, only those without default values are 'mandatory'**

```
population_data = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

def print_top_rows(input_data, n=3, m=5):                    # n=3: default value for argument n
    top_rows = input_data[:n]
    print(top_rows)

print_top_rows(population_data)                  # not input 'n' will make n use the default value 3
print_top_rows(population_data, 5)               # now n=5, not using "n=5" but just "5": positional arg
print_top_rows(population_data, n=10)            # n=10 now, 'n' is called a 'keyword argument'
print_top_rows(population_data, m=2, n=10)       # order of 'keyword arguments' can change order
```

Summary

- **List**
  - methods: append, update
  - tuple: immutable
  - looping over list & tuple
  - range() function
  - common operations on list / str / tuple
    - index, slicing
- **Dict**
  - add, remove, looping, get value
  - common: str, list, dict:
    len(), looping, 'in' operator
- **Function**
  - chaining functions: function calls a func
  - default value for input argument

Thank You