07.1_Sets_Dictionaries

October 4, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang

Content:

- Sets
- Dictionaries
- Review of Container data types

2 Standard Data Types in Python - Sets

Category of Data type	Data type	Example
Numeric, scalar	Integer	1
	Floats	1.2
	Complex	1.5+0.5j
	Booleans	True
Container	strings	"Hello World"
	List	[1, "Hello World"]
	Tuple	(1, "Hello World")
	Set	{1, "Hello World"}
	Dictionary	{1: "Hello World", 2: 100}

2.1 Sets in python

```
{1, "Hello World"}
```

- A collection of **unordered** objects
  - indexing and slicing will not work

- Set elements are **unique**. Duplicate elements are not allowed.
  - when you create a set, python will remove the "redundant"/"repeated" elements automatically
- Supports operations and concepts from set theory
- A set itself may be modified (mutable), but the elements contained in the set must be of an immutable type.

2.1.1 Questions (randomly selecting a student for each question):

- Can list be an element of a set?
- Can string be an element of a set?
- Can tuple be an element of a set?

[1]:
```
{[1,2]}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 {[1,2]}

TypeError: unhashable type: 'list'
```

[2]:
```
{(1,2)}
```

[2]:
```
{(1, 2)}
```

[3]:
```
{"s"}
```

[3]:
```
{'s'}
```

2.1.2 Creating a set

- Assignment operation: `set_a = {1,2}`
  - enclosed by curly brackets
- Function `set()`

[4]:
```
set_a = {1,2}
set_a
```

[4]:
```
{1, 2}
```

[5]:
```
type(set_a)
```

[5]:
```
set
```

[6]:
```
set_a = {1,2,1,3,4,1}
set_a
```

[6]:
```
{1, 2, 3, 4}
```

[7]:
```
s = "python"
```

[8]:
```
set(s)
```

[8]:
```
{'h', 'n', 'o', 'p', 't', 'y'}
```

[9]:
```
set_s = set(s)
set_s
```

[9]:
```
{'h', 'n', 'o', 'p', 't', 'y'}
```

[10]:
```
set_s[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 set_s[0]

TypeError: 'set' object is not subscriptable
```

[11]:
```
list(s)
```

[11]:
```
['p', 'y', 't', 'h', 'o', 'n']
```

[12]:
```
tuple(s)
```

[12]:
```
('p', 'y', 't', 'h', 'o', 'n')
```

[13]:
```
list_a = ['I', 'like', 'python']
```

[14]:
```
set(list_a)
```

[14]:
```
{'I', 'like', 'python'}
```

[15]:
```
list_b = list_a * 3
list_b
```

[15]:
```
['I', 'like', 'python', 'I', 'like', 'python', 'I', 'like', 'python']
```

[16]:
```
set(list_b)
```

[16]:
```
{'I', 'like', 'python'}
```

[17]:
```
a = 1,2,3,4
a
```

[17]:
```
(1, 2, 3, 4)
```

[18]:
```
type(a)
```

[18]:
```
tuple
```

[19]:
```
set(a)
```

[19]:
```
{1, 2, 3, 4}
```

[20]:
```
set(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[20], line 1
----> 1 set(1)

TypeError: 'int' object is not iterable
```

[21]:
```
set([1])
```

[21]:
```
{1}
```

[22]:
```
{[1]}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[22], line 1
----> 1 {[1]}

TypeError: unhashable type: 'list'
```

2.2 Grow a set with in-place methods

- `set.add()`:
  - add an item to a set
  - similar to `list.append()`
- `set.update()`:
  - Update a set with the union of itself and others.
  - the input argument should be an iterable (container data type)
  - similar to `list.extend()`

[23]:
```
set_a = {1,2,"python"}
```

[24]:
```
set_a.add(10)
```

[25]:
```
set_a
```

[25]:
```
{1, 10, 2, 'python'}
```

[26]:
```
set_a.add(10)
```

[27]:
```
set_a
```

[27]:
```
{1, 10, 2, 'python'}
```

[28]:
```
set_a = {1, 10, 2, 'python'}
set_a
```

[28]:
```
{1, 10, 2, 'python'}
```

[29]:
```
set_a.add((11, 12))
```

[30]:
```
set_a
```

[30]:
```
{(11, 12), 1, 10, 2, 'python'}
```

[31]:
```
set_a = {1, 10, 2, 'python'}
set_a
```

[31]:
```
{1, 10, 2, 'python'}
```

[32]:
```
set_a.update((11, 12))
```

[33]:
```
set_a
```

[33]:
```
{1, 10, 11, 12, 2, 'python'}
```

[34]:
```
set_a.update(1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[34], line 1
----> 1 set_a.update(1)

TypeError: 'int' object is not iterable
```

[35]:
```
set_a
```

[35]:
```
{1, 10, 11, 12, 2, 'python'}
```

[36]:
```
set_a.add({11, 12})
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[36], line 1
----> 1 set_a.add({11, 12})

TypeError: unhashable type: 'set'
```

[37]:
```
set_a = {1, 10, 2, 'python'}
```

[38]:
```
set_a.update([11, 12])
```

[39]:
```
set_a
```

[39]:
```
{1, 10, 11, 12, 2, 'python'}
```

[40]:
```
set_a.update(100)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[40], line 1
----> 1 set_a.update(100)

TypeError: 'int' object is not iterable
```

[41]:
```
set_a.update([100])
```

[42]:
```
set_a
```

[42]:
```
{1, 10, 100, 11, 12, 2, 'python'}
```

[43]:
```
set_a.add((100,))
```

[44]:
```
set_a
```

[44]:
```
{(100,), 1, 10, 100, 11, 12, 2, 'python'}
```

2.3 Shrink a set with in-place methods

- `set.remove()`:
  - reomove an item to a set
  - similar to `list.remove()`
- `set.pop()`:
  - removes and returns an arbitrary element from a set.

[45]:
```
list_a = [1, 10, 11, 12, 2, 'python']
```

[46]:
```
list_a.pop()
```

[46]:
```
'python'
```

[47]:
```
list_a
```

[47]:
```
[1, 10, 11, 12, 2]
```

[48]:
```
list_a.pop(0)
```

[48]:
```
1
```

[49]:
```
list_a
```

[49]:
```
[10, 11, 12, 2]
```

[50]:
```
set_a = {1, 10, 11, 12, 2, 'python'}
```

[51]:
```
set_a.remove(1)
```

[52]:
```
set_a
```

[52]:
```
{10, 11, 12, 2, 'python'}
```

[53]:
```
set_a.remove(3)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[53], line 1
----> 1 set_a.remove(3)

KeyError: 3
```

[54]:
```
set_a
```

[54]:
```
{10, 11, 12, 2, 'python'}
```

[55]:
```
set_a.pop()
```

[55]:
```
2
```

[56]:
```
set_a
```

[56]:
```
{10, 11, 12, 'python'}
```

[57]:
```
set_a.pop()
```

[57]:
```
'python'
```

[58]:
```
set_a
```

[58]:
```
{10, 11, 12}
```

[59]:
```
len(set_a)
```

[59]:
```
3
```

2.3.1 Mathematical operations with Sets - set theory

Check whether a set is a subset of another set

[60]:
```
group1 = set([1, 2, 3, 4, 5])
group2 = set([2,6,7])
group3 = {1,2,3}
```

[61]:
```
group2.issubset(group1)
```

[61]:
```
False
```

[62]:
```
group3.issubset(group1)
```

[62]:
```
True
```

[63]:
```
group2.issuperset(group1)
```

[63]:
```
False
```

[64]:
```
group1 = set([1, 2, 3, 4, 5])
group2 = set([2,6,7])
group3 = {1,2,3}
```

[65]:
```
group1.issubset(group3)
```

[65]:
```
False
```

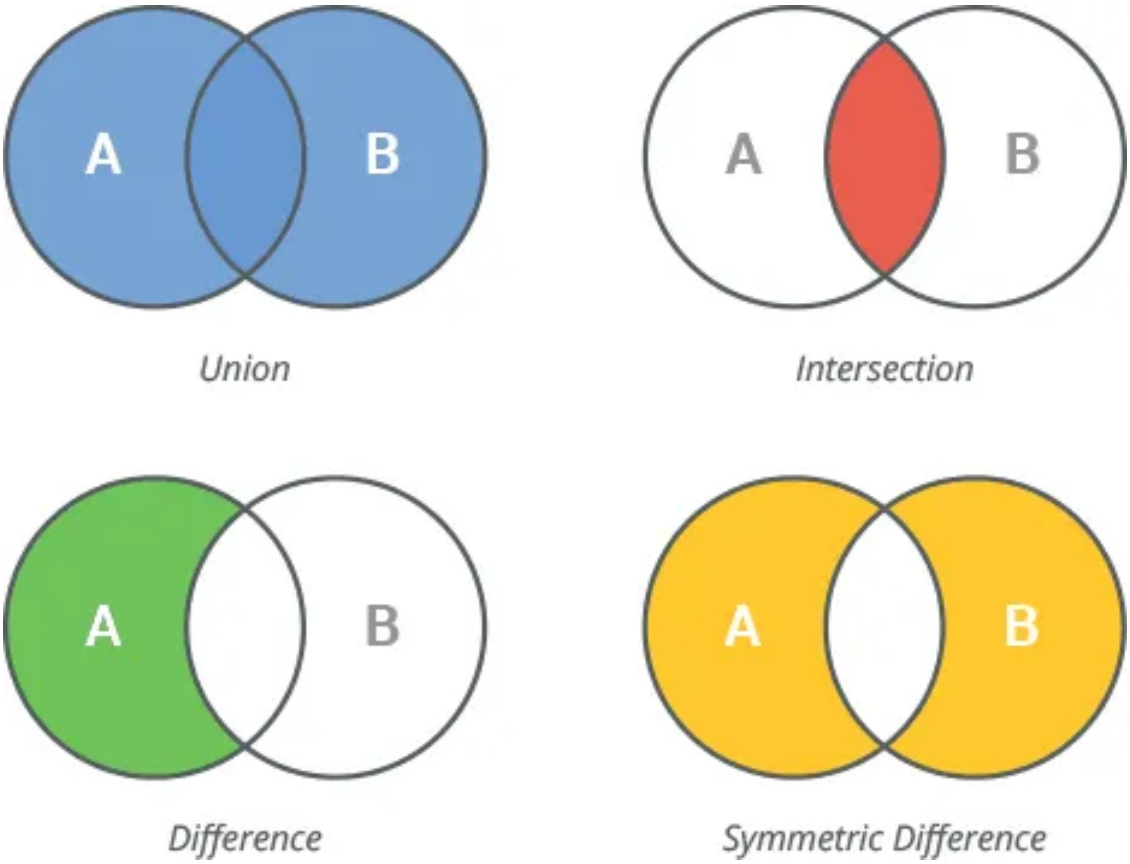
[66]:
```
group1.issubset?
```

[67]:
```
help(group1.issubset)
```

```
Help on built-in function issubset:

issubset(…) method of builtins.set instance
    Report whether another set contains this set.
```

2.3.2 More Operations on two or more sets



[68]:
```
group1 = set([1, 2, 3, 4, 5])
group2 = set([2,6,7])
group3 = {1,2,3}
```

[69]:
```
group1.union(group2)
```

[69]:
```
{1, 2, 3, 4, 5, 6, 7}
```

[70]:
```
group1.union(group2, group3)
```

[70]:
```
{1, 2, 3, 4, 5, 6, 7}
```

[71]:
```
group1.intersection(group2)
```

[71]:
```
{2}
```

[72]:
```
group1 = set([1, 2, 3, 4, 5])
group2 = set([2,6,7])
group3 = {1,2,3}
```

[73]:
```
group1.difference(group2)
```

[73]:
```
{1, 3, 4, 5}
```

[74]:
```
group2.difference(group1)
```

[74]:
```
{6, 7}
```

[75]:
```
group1.symmetric_difference(group2)
```

[75]:
```
{1, 3, 4, 5, 6, 7}
```

[76]:
```
group2.symmetric_difference(group1)
```

[76]:
```
{1, 3, 4, 5, 6, 7}
```

The `symmetric difference` of two sets is defined as the set of elements which are in one or the other, but not both, of the sets

2.3.3 Group Exercise

Write python code to remove items 10, 20, 30 from a given set `set1 = {10, 20, 30, 40, 50}`

When you are done, raise your hand!

[ ]:
```

```

[77]:
```
set1 = {10, 20, 30, 40, 50}
set1.remove(10)
set1.remove(20)
set1.remove(30)
set1
```

[77]:
```
{40, 50}
```

[78]:
```
set1 = {10, 20, 30, 40, 50}
list_toremove = [10, 20, 30]
for i in list_toremove:
    set1.remove(i)
set1
```

[78]:
```
{40, 50}
```

[79]:
```
set1 = {10, 20, 30, 40, 50}
set1.remove(10).remove(20).remove(30)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[79], line 2
      1 set1 = {10, 20, 30, 40, 50}
----> 2 set1.remove(10).remove(20).remove(30)
```

```
AttributeError: 'NoneType' object has no attribute 'remove'
```

[80]:
```
set1 = {10, 20, 30, 40, 50}
type(set1.remove(10))
```

[80]: NoneType

[81]:
```
set1 = {10, 20, 30, 40, 50}

new_set = {10, 20, 30}
```

[82]:
```
set1.difference(new_set)
```

[82]: {40, 50}

[83]:
```
set1 = {10, 20, 30, 40, 50}
```

[84]:
```
set1.remove("10")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[84], line 1
----> 1 set1.remove("10")

KeyError: '10'
```

[85]:
```
set1.remove(10)
set1.remove(20)
set1.remove(30)
set1
```

[85]: {40, 50}

[86]:
```
set1 = {10, 20, 30, 40, 50}
set1.symmetric_difference({10, 20, 30})
```

[86]: {40, 50}

[87]:
```
set1
```

[87]: {10, 20, 30, 40, 50}

3 Standard Data Types in Python - Dictionaries

Category of Data type	Data type	Example
Numeric, scalar	Integer	1
	Floats	1.2
	Complex	1.5+0.5j
	Booleans	True
Container	strings	“Hello World”
	List	[1, “Hello World”]
	Tuple	(1, “Hello World”)
	Set	{1, “Hello World”}
	Dictionary	{1: “Hello World”, 2: 100}

3.1 Dictionaries in python

```
{1: "one", 2: "two"}
```

```
{"first": "Hello World", "second": 100}
```

- Intuitive: a dictionary in python is like a dictionary we normally encounter
  - a dictionary represents a mapping from keys to values
  - e.g., a English-Spanish dictionary:
    - mapping from a English word (key) to a Spanish word (value)
- A collection of **key-value pairs**
  - key can be any immutable data type
  - value can be any data type
  - each key-value pair is considered as an item
- indexed by **keys**, not a range of numbers
  - Duplicate keys are not allowed
  - Duplicate values are allowed
- mutable
  - we can change the key-value pair using assignment statements
- insertion-ordered since python 3.7
- very powerful and efficient for some problems

3.2 Creating a dictionary

- Assignment statement: multiple ways
- Function `dict()`

Create a dictionary by defining pairs of keys and values `dict_person = {'name': 'John', 'age': '36', 'country': 'Norway'}`

- enclosed by curly brackets (similar to sets) `{}`
- comma `,` seperating items (key-value pairs)
- colon `:` seperating a key from the correspinding value

[88]:
```
dict_person = {'name': 'John', 'age': '36', 'country': 'Norway'}
dict_person
```

[88]:
```
{'name': 'John', 'age': '36', 'country': 'Norway'}
```

[89]:
```
len(dict_person)
```

[89]:
```
3
```

[90]:
```
type(dict_person)
```

[90]:
```
dict
```

Create an empty dictionary and add key-value pairs using key index assignment

```
dict_person = {}
dict_person['name'] = 'John'
```

- for dictionaries, key is the "index"
- indexing is "similar" to lists, but the "index" is not numbers, but keys (could be any immutable data types)

[91]:
```
dict_person = {}
type(dict_person)
```

[91]:
```
dict
```

[92]:
```
dict_person['name'] = 'John'
```

[93]:
```
dict_person
```

[93]:
```
{'name': 'John'}
```

[94]:
```
# dict_person['name'] = 'John'
dict_person['age'] = 36
dict_person['country'] = 'Norway'
```

[95]:
```
dict_person
```

[95]:
```
{'name': 'John', 'age': 36, 'country': 'Norway'}
```

[96]:
```
dict_person[("country", "weather")] = ('Norway', 'sunny')
```

[97]:
```
dict_person
```

[97]:
```
{'name': 'John',
 'age': 36,
 'country': 'Norway',
 ('country', 'weather'): ('Norway', 'sunny')}
```

[98]:
```
dict_person[["country", "weather"]] = ('Norway', 'sunny')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[98], line 1
----> 1 dict_person[["country", "weather"]] = ('Norway', 'sunny')

TypeError: unhashable type: 'list'
```

[99]:
```
dict_person[("country1", "weather1")] = ['Norway', 'sunny']
```

[100]:
```
dict_person
```

[100]:
```
{'name': 'John',
 'age': 36,
 'country': 'Norway',
 ('country', 'weather'): ('Norway', 'sunny'),
 ('country1', 'weather1'): ['Norway', 'sunny']}
```

Creating a dictionary with `dict()` using pairs of keys and values

- pairs could be of any ordered container data types, e.g., tuples, lists

[101]:
```
 ( ('name','John'), ('age',"36"), ('country', "Norway"))
```

[101]:
```
(('name', 'John'), ('age', '36'), ('country', 'Norway'))
```

[102]:
```
dict_person = dict( ( ('name','John'), ('age',"36"), ('country', "Norway")) )
dict_person
```

[102]:
```
{'name': 'John', 'age': '36', 'country': 'Norway'}
```

[103]:
```
tuple_pair =  ( ('name','John'), ('age',"36"), ('country', "Norway"))
dict_person = dict(tuple_pair)
dict_person
```

[103]:
```
{'name': 'John', 'age': '36', 'country': 'Norway'}
```

[104]:
```
dict_person = dict( [ ['name','John'],
                      ['age',"36"],
                      ['country', "Norway"]] )
dict_person
```

[104]:
```
{'name': 'John', 'age': '36', 'country': 'Norway'}
```

[105]:
```
list_pair =  [ ['name','John'], ['age',"36"], ['country', "Norway"]]
dict_person = dict(list_pair)
dict_person
```

[105]:
```
{'name': 'John', 'age': '36', 'country': 'Norway'}
```

3.2.1 Obtaining values using keys

- similar to Lists' indexing

[106]:
```
dict_person
```

[106]:
```
{'name': 'John', 'age': '36', 'country': 'Norway'}
```

[107]:
```
dict_person['name']
```

[107]:
```
'John'
```

[108]:
```
dict_person['John']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[108], line 1
----> 1 dict_person['John']

KeyError: 'John'
```

3.2.2 Dictionary Methods

[109]:
```
dict_person
```

[109]:
```
{'name': 'John', 'age': '36', 'country': 'Norway'}
```

[110]:
```
dict_person.keys() # returns a collection of keys
```

[110]:
```
dict_keys(['name', 'age', 'country'])
```

[111]:
```
dict_person.values() # returns a collection of values
```

[111]:
```
dict_values(['John', '36', 'Norway'])
```

`in` operator

- tells you whether something appears as a **key** in the dictionary

[112]:
```
dict_person
```

[112]:
```
{'name': 'John', 'age': '36', 'country': 'Norway'}
```

[113]:
```
'name' in dict_person
```

[113]:
```
True
```

[114]:
```
'name' in dict_person.keys()
```

[114]: True

[115]:
```
'John' in dict_person
```

[115]: False

[116]:
```
dict_person
```

[116]: {'name': 'John', 'age': '36', 'country': 'Norway'}

[117]:
```
'John' in dict_person.values()
```

[117]: True

[118]:
```
len(dict_person)
```

[118]: 3

[119]:
```
dict_person.items()
```

[119]: dict_items([('name', 'John'), ('age', '36'), ('country', 'Norway')])

Combining two dictionaries can be done with the `update` method (in-place method)

[120]:
```
dict_new = {1:3, 4:5}
dict_new
```

[120]: {1: 3, 4: 5}

[121]:
```
dict_person
```

[121]: {'name': 'John', 'age': '36', 'country': 'Norway'}

[122]:
```
dict_person.update(dict_new)
```

[123]:
```
dict_person
```

[123]: {'name': 'John', 'age': '36', 'country': 'Norway', 1: 3, 4: 5}

3.2.3 Looping dictionaries

- If you use a dictionary in a `for` statement, it traverses the `keys` of the dictionary

[124]:
```
dict_person
```

[124]: {'name': 'John', 'age': '36', 'country': 'Norway', 1: 3, 4: 5}

[125]:
```
for a in dict_person:
    print(a)
```

```
name
age
country
1
4
```

[126]:
```
dict_person
```

[126]:
```
{'name': 'John', 'age': '36', 'country': 'Norway', 1: 3, 4: 5}
```

[127]:
```
for a in dict_person.values():
    print(a)
```

```
John
36
Norway
3
5
```

[128]:
```
for a in dict_person:
    print(dict_person[a])
```

```
John
36
Norway
3
5
```

[129]:
```
dict_person
```

[129]:
```
{'name': 'John', 'age': '36', 'country': 'Norway', 1: 3, 4: 5}
```

[130]:
```
for key in dict_person:
    print(key, dict_person[key])
```

```
name John
age 36
country Norway
1 3
4 5
```

[131]:
```
dict_person.items()
```

[131]:
```
dict_items([('name', 'John'), ('age', '36'), ('country', 'Norway'), (1, 3), (4,
5)])
```

[132]:
```
dict_person
```

[132]:
```
{'name': 'John', 'age': '36', 'country': 'Norway', 1: 3, 4: 5}
```

[133]:
```
for key, v in dict_person.items(): #getting key,value pairs with method
 ↪`items()`
    print(key, v)
```

```
name John
age 36
country Norway
1 3
4 5
```

3.2.4 Group exercise

Write a Python program to convert two lists into a dictionary in a way that item from list1 is the key and item from list2 is the value

```
keys = ['Ten', 'Twenty', 'Thirty']
values = [10, 20, 30]
```

When you are done, raise your hand!

[134]:
```
keys = ['Ten', 'Twenty', 'Thirty']
values = [10, 20, 30]
```

[135]:
```
dict1 = {}
```

[136]:
```
for i in range(len(keys)):
    dict1[keys[i]] = values[i]
```

[137]:
```
dict1
```

[137]:
```
{'Ten': 10, 'Twenty': 20, 'Thirty': 30}
```

[138]:
```
keys = ['Ten', 'Twenty', 'Thirty']
values = [10, 20, 30, 40]
dict1 = {}
for i in range(len(keys)):
    dict1[keys[i]] = values[i]
dict1
```

[138]:
```
{'Ten': 10, 'Twenty': 20, 'Thirty': 30}
```

[139]:
```
keys = ['Ten', 'Twenty', 'Thirty']
values = [10, 20, 30, 40]
dict1 = {}
for i in range(len(values)):
    dict1[keys[i]] = values[i]
dict1
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[139], line 5
      3 dict1 = {}
      4 for i in range(len(values)):
----> 5     dict1[keys[i]] = values[i]
      6 dict1

IndexError: list index out of range
```

[140]:
```
keys = ['Ten', 'Twenty', 'Thirty']
values = [10, 20, 30]
```

[141]:
```
[['Ten', 10], ['Twenty', 10], ['Thirty', 30]]
```

[141]:
```
[['Ten', 10], ['Twenty', 10], ['Thirty', 30]]
```

[142]:
```
dict([['Ten', 10], ['Twenty', 10], ['Thirty', 30]])
```

[142]:
```
{'Ten': 10, 'Twenty': 10, 'Thirty': 30}
```

[143]:
```
[[keys[0], values[0]], [keys[1], values[1] ], [keys[2], values[2]]]
```

[143]:
```
[['Ten', 10], ['Twenty', 20], ['Thirty', 30]]
```

[144]:
```
list_pair = []
for i in range(3):
    list_pair.append([keys[i], values[i]])
dict(list_pair)
```

[144]:
```
{'Ten': 10, 'Twenty': 20, 'Thirty': 30}
```

[145]:
```
list(zip(keys, values))
```

[145]:
```
[('Ten', 10), ('Twenty', 20), ('Thirty', 30)]
```

[146]:
```
dict(zip(keys, values))
```

[146]:
```
{'Ten': 10, 'Twenty': 20, 'Thirty': 30}
```

`zip()` function takes two or more iterables (like list, dict, string), aggregates them in a tuple, and returns it.

4 Review of Python Container Data Types

Category of Data type	Data type	Example	Ordered	Mutable	Unique
Container	strings	“Hello World”			
	List	[1, “Hello World”]			
	Tuple	(1, “Hello World”)			
	Set	{1, “Hello World”}			
	Dictionary	{1: “Hello World”, 2: 100}			

4.0.1 *Questions?*

- What is a string in python?
- What is a list in python?
- What is a tuple in python?
- What is a set in python?
- What is a dictionary in python?
- What are the differences between these container data types?

[147]:
```
{[1,2], 2}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[147], line 1
----> 1 {[1,2], 2}

TypeError: unhashable type: 'list'
```

4.1 Review of Python Container Data Types

Category of Data type	Data type	Example	Ordered	Mutable	Unique
Container	strings	“Hello World”	Yes	No	No
	List	[1, “Hello World”]	Yes	Yes	No
	Tuple	(1, “Hello World”)	Yes	No	No
	Set	{1, “Hello World”}	No	No	Yes

Category of Data type	Data type	Example	Ordered	Mutable	Unique
	Dictionary	{1: “Hello World”, 2: 100}	insertion-ordered	Yes	Yes

4.2 Further readings

- tutorial of python Sets
- tutorial of python dictionaries

5 Assginments

- HW5 released later today, due by 10/11

6 Next Class

- Scripts and modules
- Python Ecosystem

[ ]: