Tuples

Table of Contents

Creating a `tuple`

- We have two basic ways to create a tuple.

```
()
tuple()
```

Creating a `tuple`

- A `tuple` can be created by enclosing values, separated by commas, in parentheses 👉 `()`.
- Another way to create a `tuple` is to call the `tuple()` function.

- `()`
- `tuple()`

```
tuple_1 = ('h', 'a', 'p', 'p', 'y')
word = 'happy'
tuple_2 = tuple(word)
print(tuple_1)
print(tuple_2)
```

⚠️ an iterable object can be converted into a `tuple`

```
('h', 'a', 'p', 'p', 'y')
('h', 'a', 'p', 'p', 'y')
```

Creating a `tuple`

- Single element tuple :

```
my_tuple = ('Solar')
print(my_tuple, type(my_tuple), sep='\n')

my_tuple = ('Solar',)
print(my_tuple, type(my_tuple), sep='\n')
```

Output

```
Solar
<class 'str'>
```

```
('Solar',)
<class 'tuple'>
```

Creating a `tuple`

- `list` to `tuple`, `tuple` to `list`

```
1  my_tuple=(1, 4, 3, 4, 5, 6, 7, 4)
2
3  my_list = list(my_tuple)
4
5  print(type(my_list), my_list)
6
```

```
1  <class 'list'> [1, 4, 3, 4, 5, 6, 7, 4]
2
```

```
1  my_list = [1, 4, 3, 4, 5, 6, 7, 4]
2
3  my_tuple = tuple(my_list)
4
5  print(type(my_tuple), my_tuple)
6
```

```
1  <class 'tuple'> (1, 4, 3, 4, 5, 6, 7, 4)
2
```

How can We Use a Tuple?

How can We Use a `tuple`?

- (..Continued) (review of the pre-class)
  - Just like the **`lists`**, the **`tuples`** support indexing :

```
1  even_no = (0, 2, 4)
2  print(even_no[0])
3  print(even_no[1])
4  print(even_no[2])
5  print(even_no[3])
6
```

How can We Use a `tuple`?

- (..Continued)(review of the pre-class)
  - `tuple` is **immutable**.

```
1  city_list = ['Tokyo', 'Istanbul', 'Moskow', 'Dublin']
2
3  city_tuple = tuple(city_list)
4
5  city_tuple[0] = 'New York'  # you can't assign a value
6
```

How can We Use a `tuple`?

- (..Continued)(review of the pre-class)
  - And one of the most important differences of **`tuple`**s from **`list`**s is that **`tuple`** object does not support item assignment. Yes, because **`tuple`** is immutable.

```
1  city_list = ['Tokyo', 'Istanbul', 'Moskow', 'Dublin']
2
3  city_tuple = tuple(city_list)
4
5  city_tuple[0] = 'New York'  # you can't assign a value
6
```

```
1  ---------------------------------------------------------------------------
2  TypeError: 'tuple' object does not support item assignment
3
```

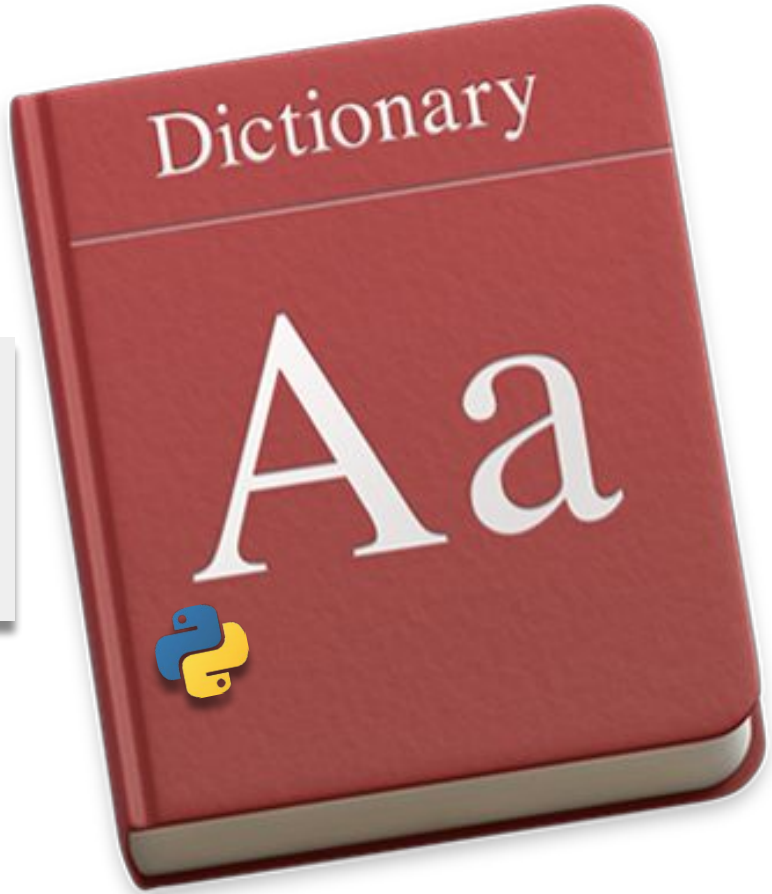
Dictionaries

Table of Contents

Definitions

- Dictionaries

```
{key1 : value1,
 key2 : value2}
```



Creating a `dict` (review)

- We have two basic ways to create a dictionary.

- `{}`
- `dict()`

Creating a `dict` (review)

- A `dict` can be created by enclosing pairs, separated by commas, in curly-braces 👉 `{}`.
- Another way to create a `dict` is to call the `dict()` function.

- `{}`
- `dict()`

```
grocer1 = {'fruit':'apple', 'drink':'water'}
grocer2 = dict(fruit='apple', drink='water')
print(grocer1)
print(grocer2)
```

```
{'fruit': 'apple', 'drink': 'water'}
{'fruit': 'apple', 'drink': 'water'}
```

Creating a `dict`

- Assigning a value to a key

```
state_capitals = {'Arkansas': 'Little Rock',
                  'Colorado': 'Denver',
                  'California': 'Sacramento',
                  'Georgia': 'Atlanta'
                 }

print(state_capitals['Colorado']) # accessing method

```

```
Denver

```

Creating a `dict`

- Let's add a new item into the `dict`.

```
1  state_capitals = {'Arkansas': 'Little Rock',
2                    'Colorado': 'Denver',
3                    'California': 'Sacramento',
4                    'Georgia': 'Atlanta'
5                   }
6
7  state_capitals['Virginia'] = 'Richmond' # adding a new item
8
9  print(state_capitals)
10
```

Creating a `dict`

- Let's add a new item into the `dict`.

```
state_capitals = {'Arkansas': 'Little Rock',
                  'Colorado': 'Denver',
                  'California': 'Sacramento',
                  'Georgia': 'Atlanta'
                 }

state_capitals['Virginia'] = 'Richmond' # adding a new item

print(state_capitals)

```

```
{'Arkansas': 'Little Rock',
'Colorado': 'Denver',
'California': 'Sacramento',
'Georgia': 'Atlanta',
'Virginia': 'Richmond'}

```

Creating a `dict`

Tips:

- Note that keys and values can be of different types.

```
1  mix_values = {'animal': ('dog', 'cat'),  # tuple type
2                'planet': ['Neptun', 'Saturn', 'Jupiter'],  # list type
3                'number': 40,  # int type
4                'pi': 3.14,  # float type
5                'is_good': True}  # bool type
6
7  mix_keys = {22 : "integer",
8              1.2 : "float",
9              True : "boolean",
10             "key" : "string"}
11
```

Creating a `dict`

- Now, it's time to create a `dict` using `dict()` function :

```
1  dict_by_dict = dict(animal='dog', planet='neptun', number=40, pi=3.14, is_good=True)
2
3  print(dict_by_dict)
4
```

```
1  {'animal': 'dog',
2  'planet': 'neptun',
3  'number': 40,
4  'pi': 3.14,
5  'is_good': True}
6
```

⚠ **Avoid ! :**

- Do not use quotes for `keys` when using the `dict()` function to create a dictionary.

Main Operations with Dictionaries

Main Operations with `dicts` (review)

- You can access all;
  - items using the `.items()` method,
  - `keys` using the `.keys()` method,
  - `values` using the `.values()` method.

Main Operations with `dicts` (review)

- Let's take a look at this example :

```
1  dict_by_dict = {'animal': 'dog',
2                  'planet': 'neptun',
3                  'number': 40,
4                  'pi': 3.14,
5                  'is_good': True}
6
7  print(dict_by_dict.items(), '\n')
8  print(dict_by_dict.keys(), '\n')
9  print(dict_by_dict.values())
10
```

```
1  dict_items([('animal', 'dog'), ('planet', 'neptun'),
2              ('number', 40), ('pi', 3.14), ('is_good', True)])
3
4  dict_keys(['animal', 'planet', 'number', 'pi', 'is_good'])
5
6  dict_values(['dog', 'neptun', 40, 3.14, True])
7
```

Main Operations with `dicts` (review)

- Another way to add a new item into a `dict` is the `.update()` method.

```
1  dict_by_dict = {'animal': 'dog',
2                  'planet': 'neptun',
3                  'number': 40,
4                  'pi': 3.14,
5                  'is_good': True}
6
7  dict_by_dict.update({'is_bad': False})
8
9  print(dict_by_dict)
10
```

```
1  {'animal': 'dog',
2  'planet': 'neptun',
3  'number': 40,
4  'pi': 3.14,
5  'is_good': True,
6  'is_bad': False}
7
```

Main Operations with `dicts` (review)

- Python allows us to remove an item from a `dict` using the `del` function.

The formula syntax is : `del dictionary_name['key']`

```
1  dict_by_dict = {'animal': 'dog',
2                  'planet': 'neptun',
3                  'number': 40,
4                  'pi': 3.14,
5                  'is_good': True,
6                  'is_bad': False}
7
8  del dict_by_dict['animal']
9
10 print(dict_by_dict)
11
```

```
1 {'planet': 'neptun',
2 'number': 40,
3 'pi': 3.14,
4 'is_good': True,
5 'is_bad': False}
6
```

Nested Dictionaries

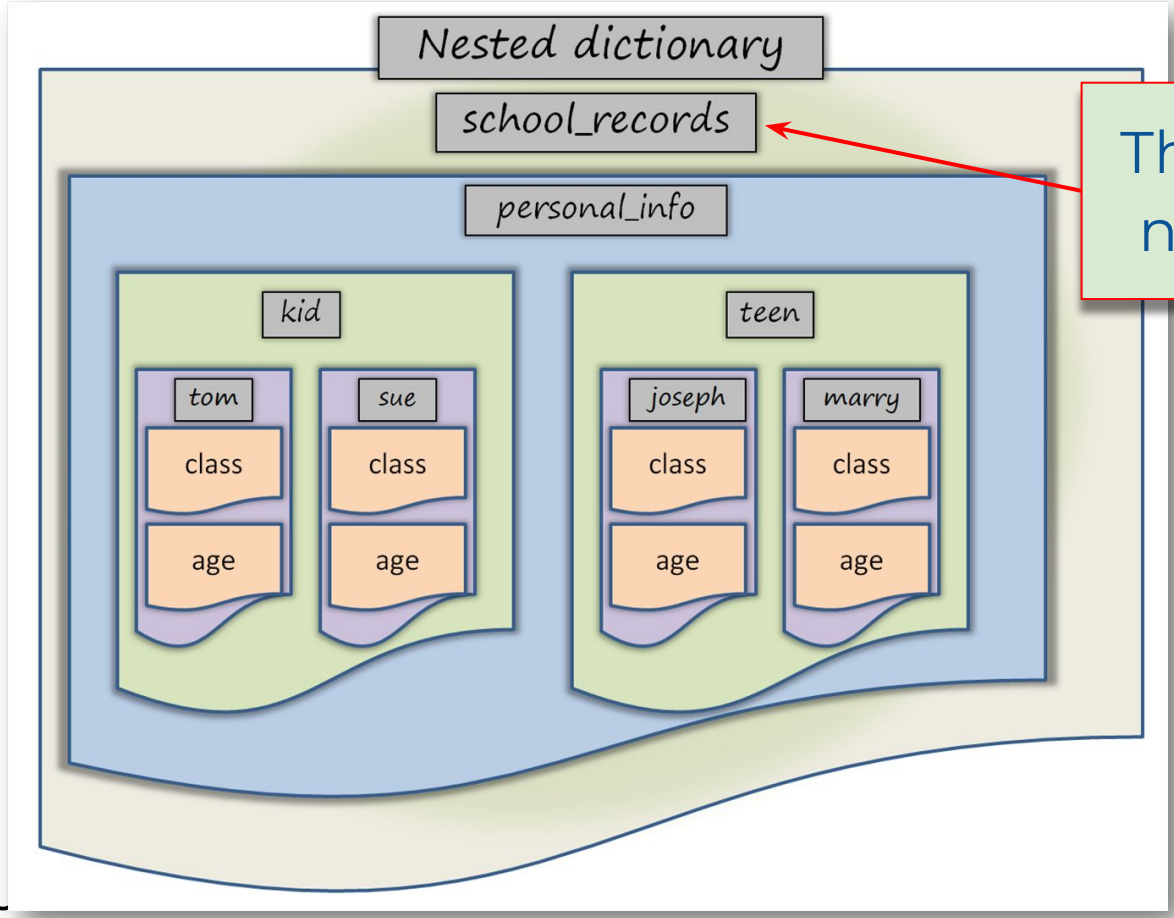
Nested `dicts` (review pre-class)

- In some cases you need to work with nested `dict`. Consider the following pre-class example :

```
school_records={
    "personal_info":
        {"kid":{"tom": {"class": "intermediate", "age": 10},
                "sue": {"class": "elementary", "age": 8}
               },
         "teen":{"joseph":{"class": "college", "age": 19},
                 "marry":{"class": "high school", "age": 16}
                },
        },

    "grades_info":
        {"kid":{"tom": {"math": 88, "speech": 69},
                "sue": {"math": 90, "speech": 81}
               },
         "teen":{"joseph":{"coding": 80, "math": 89},
                 "marry":{"coding": 70, "math": 96}
                },
        },
}
```

Nested `dicts` (review pre-class)



The first part of the nested dictionary.

Nested `dicts` (review pre-class)

- You can use traditional accessing method - square brackets - also in the nested dictionaries.

```
school_records={
    "personal_info":
        {"kid":{"tom": {"class":"intermediate", "age":10},
                "sue": {"class":"elementary", "age":8}
               },
         "teen":{"joseph":{"class":"college", "age":19},
                 "marry":{"class":"high school", "age":16}
                },
        },
}

print(school_records['personal_info']['teen']['marry']['age'])
```

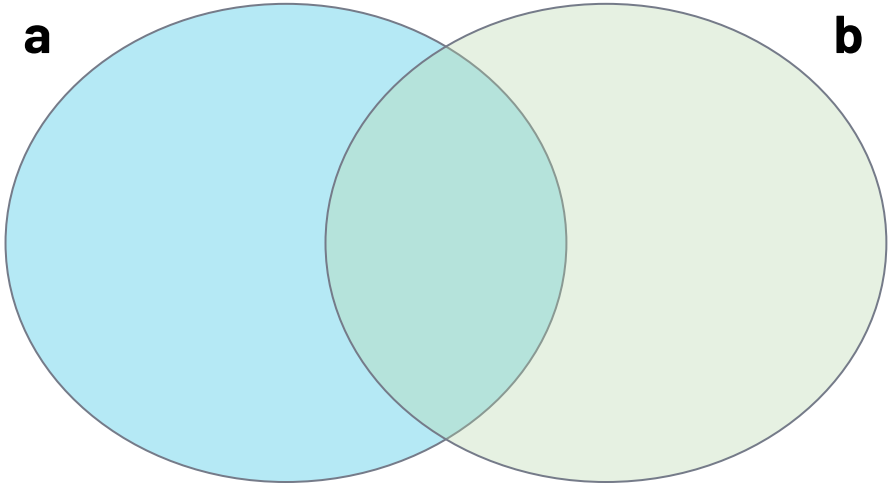
```
16
```

Sets

Table of Contents

Definitions

- No repetition
- Math operations
  - union
  - intersection
  - difference
- Unordered elements



Creating a `set`

- A `set` can be created by enclosing values, separated by commas, in curly braces 👉 `{}`.
- Another way to create a `set` is to call the `set()` function.

- `{}`
- `set()`

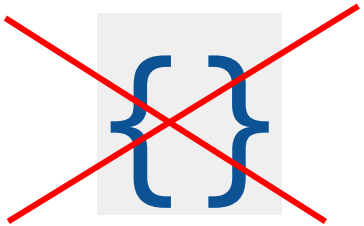
```
set_1 = {'red', 'blue', 'pink', 'red'}
colors = 'red', 'blue', 'pink', 'red'
set_2 = set(colors)
print(set_1)
print(set_2)
```

```
{'red', 'blue', 'pink'}
{'red', 'blue', 'pink'}
```

⚠️ different order from the previous slide

Creating a `set`

- Creating an empty set



To create an empty `set`, you **can not** use 👉 `{}`. The only way to create an empty set is `set()` function.

Creating a `set` (review of pre-class)

```
1  flower_list = ['rose', 'violet', 'carnation', 'rose', 'orchid', 'rose', 'orchid']
2  flowerset = set(flower_list)
3  flowerlist = list(flowerset)
4
5  print(flowerset)
6  print(flowerlist)
7
```

```
1  {'orchid', 'carnation', 'violet', 'rose'}
2  ['orchid', 'carnation', 'violet', 'rose']
3
```

Main Operations with Sets

Main Operations with `sets` (review)

- The methods that can be used with `sets` :

- `.add()` : Adds a new item to the set.
- `.remove()` : Allows us to delete an item.
- `.intersection()` : Returns the intersection of two sets.
- `.union()` : Returns the unification of two sets.
- `.difference()` : Gets the difference of two sets.

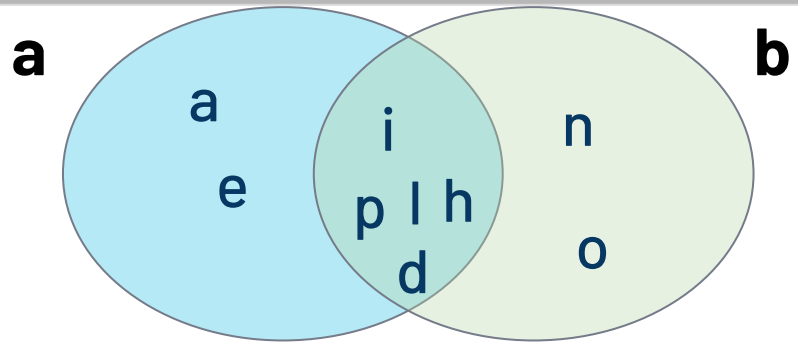
Main Operations with sets

- Let's take a look these two sets below :

```
a = set('philadelphia')
print(a)
b = set('dolphin')
print(b)
```

```
{'a', 'e', 'i', 'd', 'l', 'p', 'h'}
```

```
{'d', 'l', 'o', 'p', 'n', 'i', 'h'}
```



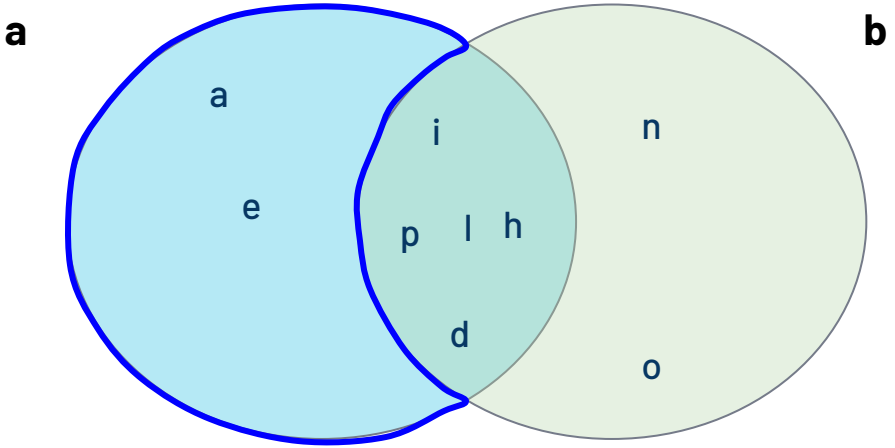
Main Operations with `sets`

- Basic `set` operations :

`.difference(arg)`

```
print(a - b)
print(a.difference(b))
```

```
{'a', 'e'}
```



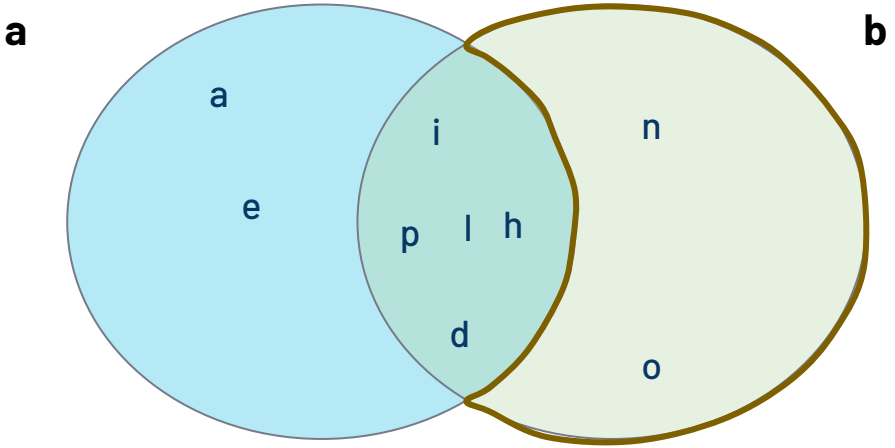
Main Operations with `sets`

- Basic `set` operations :

`.difference(arg)`

```
print(b - a)
print(b.difference(a))
```

```
{'n', 'o'}
```



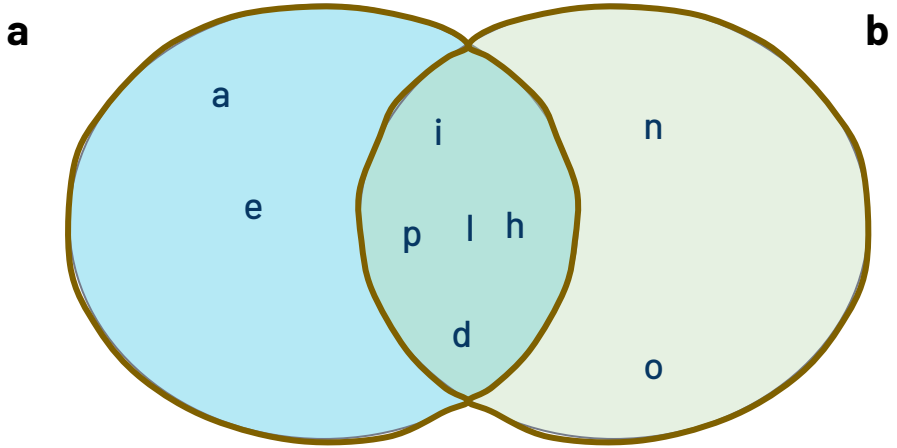
Main Operations with `sets`

- Basic `set` operations :

`.union(arg)`

```
print(a | b)
print(a.union(b))
```

```
{'p', 'h', 'i', 'l', 'd', 'o', 'n', 'a', 'e'}
```



Main Operations with `sets`

- Basic `set` operations :

`.intersection(arg)`

```
print(a & b)
print(a.intersection(b))
```

```
{'p', 'h', 'i', 'l', 'd'}
```