



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
```

What is the output? Try to figure out in your mind...



Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

3

► 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



▶ Task 🙋

- ▶ Access and print the **items**, **keys** and **values** of the same **family dict** you created.
- ▶ Note : Get the output of the above as a **list** type.

▶ Main Operations with dicts



- ▶ The code can be like :

```
print(list(family.items()), "\n")
print(list(family.keys()), "\n")
print(list(family.values()))
```

```
[('name1', 'Joseph'), ('name2', 'Bella'), ('name3', 'Aisha'), ('name4', 'Tom')]
['name1', 'name2', 'name3', 'name4']
['Joseph', 'Bella', 'Aisha', 'Tom']
```

▶ Main Operations with dicts (review) ▶

- ▶ `.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
```

▶ 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

► Task 🖐️

- Add a new family member name to the dictionary you created using `.update()` method.



Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

9

► Main Operations with dicts



- The code can be like :

```
family = {'name1': 'Joseph',  
          'name2': 'Bella',  
          'name3': 'Aisha',  
          'name4': 'Tom'}  
family.update({'name5': 'Alfred'})  
print(family)
```

```
family = {'name1': 'Joseph',  
          'name2': 'Bella',  
          'name3': 'Aisha',  
          'name4': 'Tom',  
          'name5': 'Alfred'}  
}
```

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
```

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
```

► Main Operations with dicts



► Task 🖱️

- Remove the female members from the **dict** using **del** operator.

► Main Operations with dicts



- The code can be like :

```
del family['name2']  
del family['name3']  
  
print(family)
```

```
family = {'name1': 'Joseph',  
          'name4': 'Tom',  
          'name5': 'Alfred',  
          }
```



► Main Operations with dicts

- The code can be like :

```
del family['name2']  
del family['name3']  
  
print(family)
```

Can you do the same thing in a single line ?

```
family = {'name1': 'Joseph',  
          'name4': 'Tom',  
          'name5': 'Alfred'  
}
```



Pear Deck Interactive Slide
Do not remove this bar

15

► Main Operations with dicts



- The code can be like :

```
del family['name2']  
del family['name3']  
  
print(family)
```

Option-1

```
del family['name2'], family['name3']  
  
print(family)
```

Option-2

```
family = {'name1': 'Joseph',  
          'name4': 'Tom',  
          'name5': 'Alfred'  
}
```


► Main Operations with dicts (review) ►

Using the `in` and the `not in` operator, you can check if the `key` is in the `dictionary`.

- When we use the `in` operator; if the `key` is in the dictionary, the result will be `True` otherwise `False`.
- When we use the `not in`; if the `key` is not in the dictionary, the result will be `True` otherwise `False`.

► Main Operations with dicts (review) ►

Using the `in` and the `not in` operator, you can check if the `key` is in the `dictionary`.

- When we use the `in` operator; if the `key` is in the dictionary, the result will be `True` otherwise `False`.
- When we use the `not in`; if the `key` is not in the dictionary, the result will be `True` otherwise `False`.

```
1 dict_by_dict = {'planet': 'neptun',  
2               'number': 40,  
3               'pi': 3.14,  
4               'is_good': True,  
5               'is_bad': False}  
6  
7 print('pi' in dict_by_dict)  
8 print('animal' not in dict_by_dict) # remember, we have deleted 'animal'  
9
```

Main Operations with dicts (review)

Using the `in` and the `not in` operator, you can check if the `key` is in the `dict`ionary.

- When we use the `in` operator; if the `key` is in the dictionary, the result will be `True` otherwise `False`.
- When we use the `not in`; if the `key` is not in the dictionary, the result will be `True` otherwise `False`.

```
1 dict_by_dict = {'planet': 'neptun',  
2                 'number': 40,  
3                 'pi': 3.14,  
4                 'is_good': True,  
5                 'is_bad': False}  
6  
7 print('pi' in dict_by_dict)  
8 print('animal' not in dict_by_dict) # remember, we have deleted 'animal'
```

```
1 True  
2 True  
3
```

Main Operations with dicts

Task

- ▶ Check the “Aisha” if she is in the `dict` using `in` operator.



▶ Main Operations with dicts

- ▶ The code can be like :

```
print('name3' in family)
```

```
False
```



▶ Nested Dictionaries

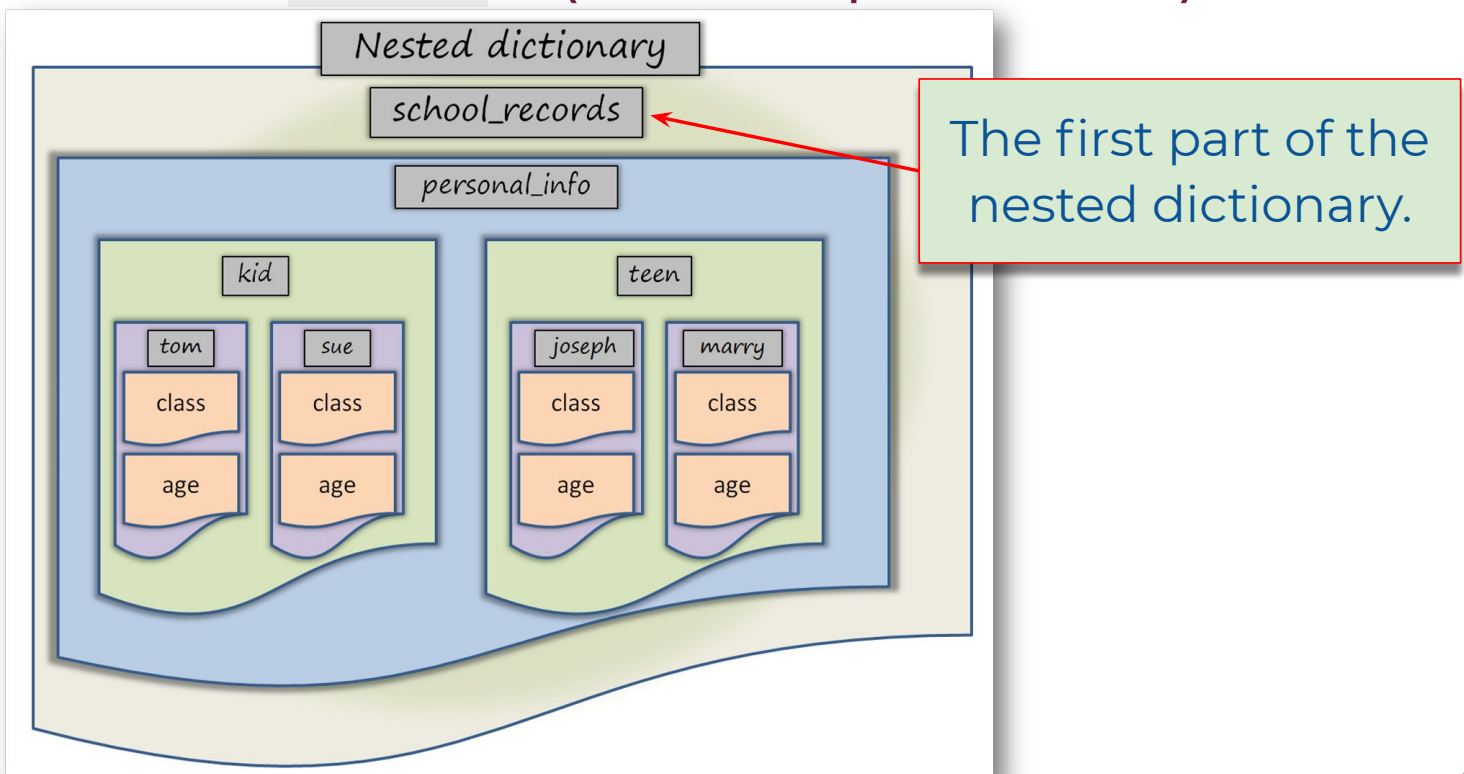
Nested dicts (review pre-class)

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

```
1 school_records={
2     "personal_info":
3         {"kid": {"tom": {"class": "intermediate", "age": 10},
4                    "sue": {"class": "elementary", "age": 8}
5          },
6         "teen": {"joseph": {"class": "college", "age": 19},
7                  "marry": {"class": "high school", "age": 16}
8          },
9     },
10
11    "grades_info":
12        {"kid": {"tom": {"math": 88, "speech": 69},
13                  "sue": {"math": 90, "speech": 81}
14          },
15        "teen": {"joseph": {"coding": 80, "math": 89},
16                  "marry": {"coding": 70, "math": 96}
17          },
18    },
19 }
20
```

23

Nested dicts (review pre-class)



24



Nested dicts (review pre-class)

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

```
1 school_records={
2     "personal_info":
3         {"kid":{"tom":{"class":"intermediate", "age":10},
4                 "sue":{"class":"elementary", "age":8}
5             }},
6     "teen":{"joseph":{"class":"college", "age":19},
7            "marry":{"class":"high school", "age":16}
8            },
9 },
10 }
11
12 print(school_records['personal_info']['teen']['marry']['age'])
13
```



Nested dicts (review pre-class)

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

```
1 school_records={
2     "personal_info":
3         {"kid":{"tom":{"class":"intermediate", "age":10},
4                 "sue":{"class":"elementary", "age":8}
5             }},
6     "teen":{"joseph":{"class":"college", "age":19},
7            "marry":{"class":"high school", "age":16}
8            },
9 },
10 }
11
12 print(school_records['personal_info']['teen']['marry']['age'])
13
```

```
1 16
2
```

Nested dicts

- **Task** : Access and print the exams and their grades of Joseph as in two types; one is a **list** form and one is a **dict**.

```
1 school_records={
2     "personal_info":
3         {"kid":{"tom": {"class": "intermediate", "age": 10},
4                 "sue": {"class": "elementary", "age": 8}
5             },
6         "teen":{"joseph":{"class": "college", "age": 19},
7                 "marry":{"class": "high school", "age": 16}
8             },
9     },
10
11     "grades_info":
12         {"kid":{"tom": {"math": 88, "speech": 69},
13                 "sue": {"math": 90, "speech": 81}
14             },
15         "teen":{"joseph":{"coding": 80, "math": 89},
16                 "marry":{"coding": 70, "math": 96}
17             },
18     },
19 }
```

Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

27

Nested dicts

- The code can be like :

```
1 school_records={
2     "personal_info":
3         {"kid":{"tom": {"class": "intermediate", "age": 10},
4                 "sue": {"class": "elementary", "age": 8}
5             },
6         "teen":{"joseph":{"class": "college", "age": 19},
7                 "marry":{"class": "high school", "age": 16}
8             },
9     },
10
11     "grades_info":
12         {"kid":{"tom": {"math": 88, "speech": 69},
13                 "sue": {"math": 90, "speech": 81}
14             },
15         "teen":{"joseph":{"coding": 80, "math": 89},
16                 "marry":{"coding": 70, "math": 96}
17             },
18     },
19 }
20 print(list(school_records["grades_info"]["teen"]["joseph"].items()))
21 print(school_records["grades_info"]["teen"]["joseph"])
22
```

Output

```
[('coding', 80), ('math', 89)]
{'coding': 80, 'math': 89}
```


28



Nested dicts

Task

- Let's create and print a **dict** (named **friends**) which consists of **first** and **last** names of your friends.
- Each person should have first and last names.
- For example;
friend1: (*first* : *Sue*, *last* : *Bold*)
friend2: (*first* : *Steve*, *last* : *Smith*)
•
•

Create using curly braces  {}



Nested dicts

- The code can be like :

```
1 friends = {  
2     "friend1" : {"first" : "Sue", "last" : "Bold"},  
3     "friend2" : {"first" : "Steve", "last" : "Smith"},  
4     "friend3" : {"first" : "Sergio", "last" : "Tatoo"}  
5 }  
6 print(friends)  
7
```

Nested dicts

Create using curly braces 🙌 {}

Task 🙌

- ▶ Let's create and print a **dict** (named **favourite**) which consists of first and last names of your **friends** and **family** members.
- ▶ Each person should have first and last names and the groups (friends and family) have three person each.
- ▶ **For** example;

friends :

friend1: (first : Sue, last : Bold)

family :

family1: (first : Steve, last : Smith)

Nested dicts

- ▶ The code can be like :

```
1 favourite = {
2     "friends" : {
3         "friend1" : {"first" : "Sue", "last" : "Bold"},
4         "friend2" : {"first" : "Steve", "last" : "Smith"},
5         "friend3" : {"first" : "Sergio", "last" : "Tatoo"}
6     },
7     "family" : {
8         "family1" : {"first" : "Mary", "last" : "Tisa"},
9         "family2" : {"first" : "Samuel", "last" : "Brown"},
10        "family3" : {"first" : "Tom", "last" : "Happy"}
11    }
12 }
13 print(favourite)
14
```




Sets



CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents



- ▶ Definitions
- ▶ Creating a Set
- ▶ Main Operations with Sets

CLARUSWAY©
WAY TO REINVENT YOURSELF



Definitions

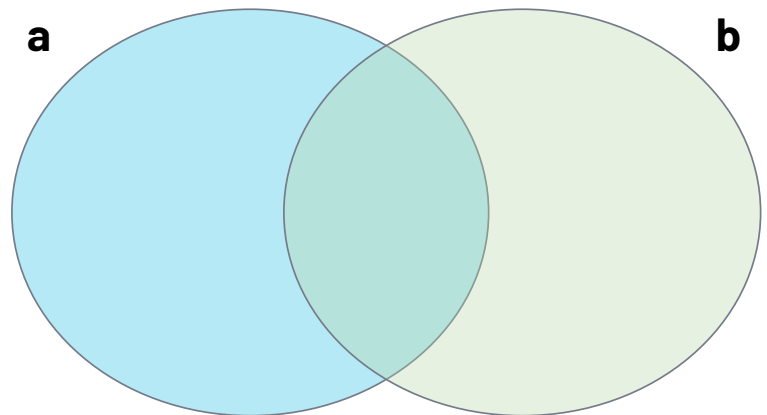
```
fruit = {'Apple', 'Orange', 'Banana'}
```

set()

Definitions



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



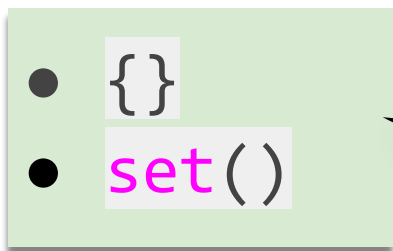


Creating a set

Creating a set



- ▶ We have two basic ways to create a set.

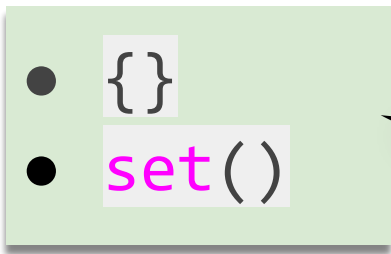


```
set_1 = {'red', 'blue', 'pink', 'red'}  
colors = 'red', 'blue', 'pink', 'red'  
set_2 = set(colors)  
print(set_1)
```

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

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_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

39

Creating a set(review of pre-class)

- ▶ Here is an example of creating an empty **set**:

input :

```
1 empty_set = set()
2
3 print(type(empty_set))
4
```

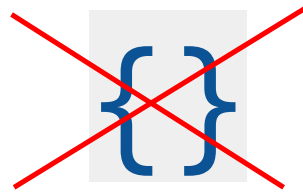
output :

```
1 <class 'set'>
2
```

40

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
```

What is the output? Try to figure out in your mind...

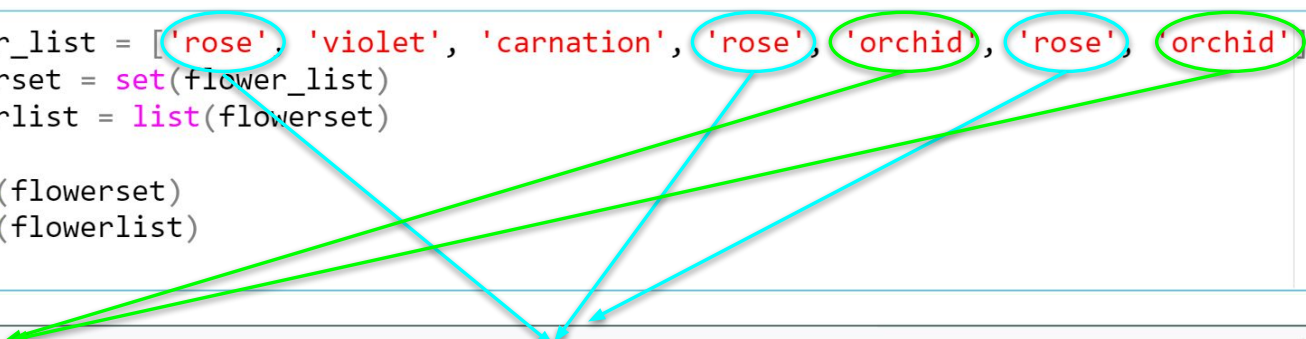


Creating a set (review of pre-class)



```
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
```



Creating a set (review of pre-class)



Task:

- Do these two sets give the same output and why?

```
a = {'carnation', 'orchid', 'rose', 'violet'}
```



```
b = {'rose', 'orchid', 'rose', 'violet', 'carnation'}
```

▶ Creating a set



- ▶ The Answer is : **True**

```
{'carnation', 'orchid', 'rose', 'violet'}
```

```
{'rose', 'orchid', 'rose', 'violet', 'carnation'}
```



▶ Main Operations with Sets

► Main Operations with sets (review)



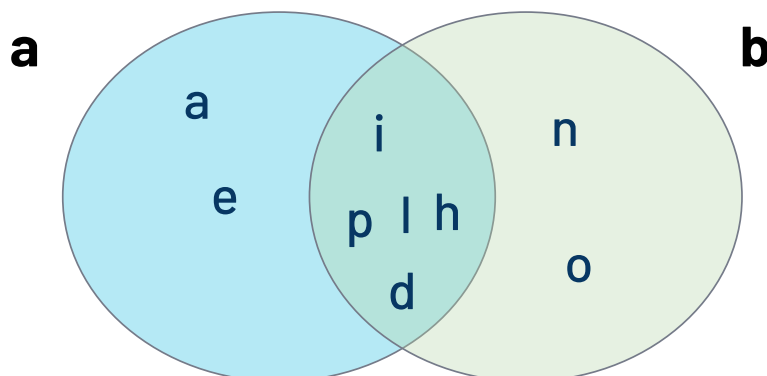
- The methods that can be used with `sets` :
 - `.add()` : Adds a new item to the set.
 - `.remove()` : Allows us to delete an item.
 - `.intersection()` : Returns the intersection of two sets.
 - `.union()` : Returns the unification of two sets.
 - `.difference()` : Gets the difference of two sets.

► Main Operations with sets



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

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



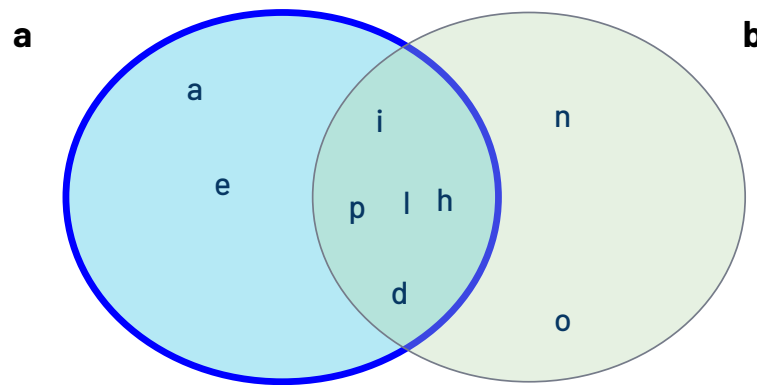


Main Operations with sets

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

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

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

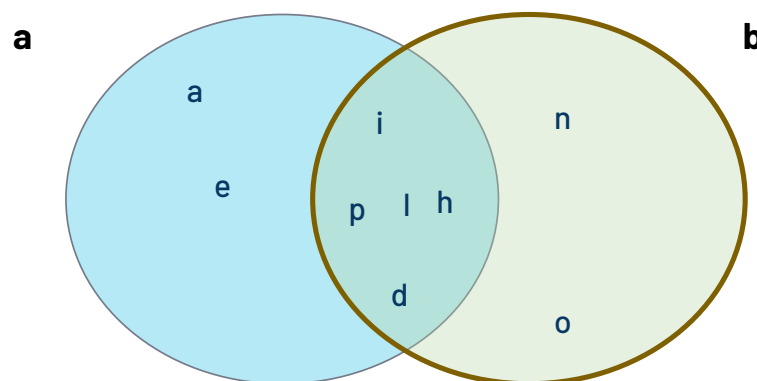


Main Operations with sets

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

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

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



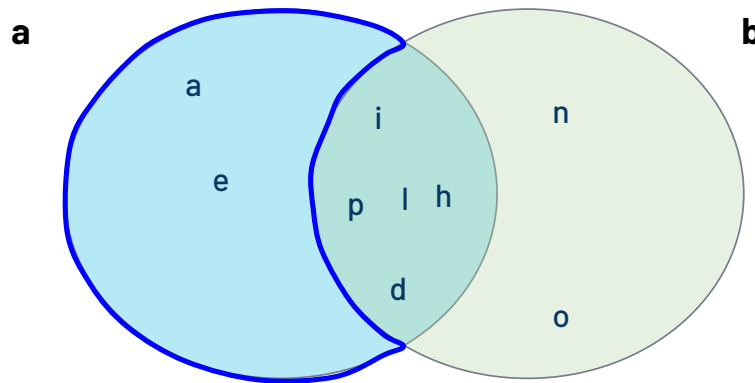
Main Operations with sets

- Basic **set** operations :

`.difference(arg)`

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

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



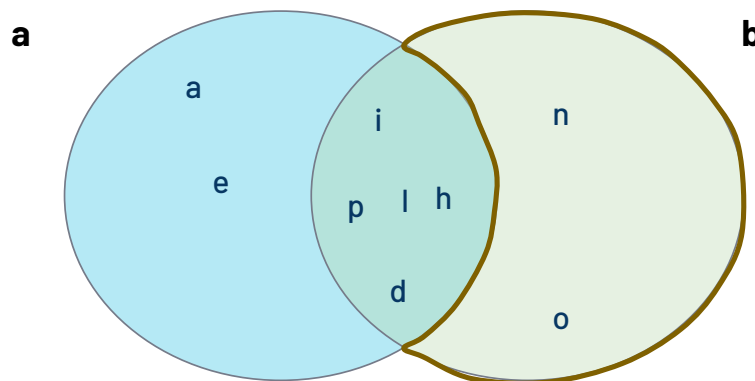
Main Operations with sets

- Basic **set** operations :

`.difference(arg)`

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

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



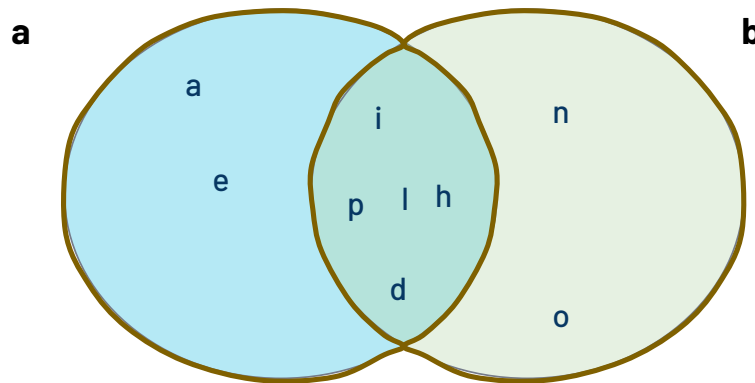
Main Operations with sets

- Basic **set** operations :

.union(arg)

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

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



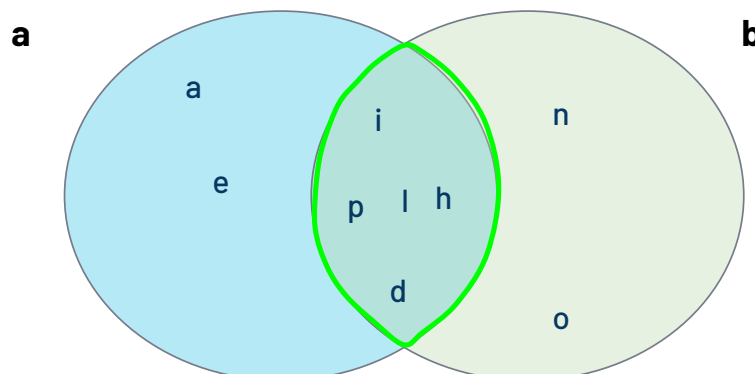
Main Operations with sets

- Basic **set** operations :

.intersection(arg)

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

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





Creating a set

Task :

- Let's create a **set** from which **str** type of the current date?
- Date style would be "mm/dd/yyyy".
- Creating a **set**, use both **set()** function and **{}** then figure out the results.



Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

55

Creating a set



The solution:

```
a = set('09/01/2021')
b = {'09/01/2021'}
print(a)
print(b)
```

```
{'1', '0', '9', '2', '/'}
{'09/01/2021'}
```



▶ Creating a set

▶ Task :

Given a `list`, create a `set` to select and print the **unique** elements of the it.

```
given_list = [1, 2, 3, 3, 3, 3, 4, 4, 5, 5]
```

▶ Creating a set



▶ The code might be like :

```
given_list = [1, 2, 3, 3, 3, 3, 4, 4, 5, 5]
```

```
unique = set(given_list)
```

```
print(unique)
```

```
{1, 2, 3, 4, 5}
```

Discuss in-class! Could you do the same thing using only curly braces `{}` instead of `set()` function?



▶ Creating a set

▶ Task :

- Create two sets of string data from the capitals of the **USA** and **New Zealand**. (e.g: 'Madrid' → convert into a set)
- Perform all set operations.
 - Intersection
 - Union
 - Difference

▶ Creating a set



▶ The code might be like :

```
usa_capt = set('Washington')
nz_capt = set('Wellington')

print(usa_capt)
print(nz_capt)
```

```
{'h', 'W', 'a', 'o', 's', 'n', 'g', 'i', 't'}
{'W', 'o', 'l', 'e', 'n', 'g', 'i', 't'}
```



▶ Creating a set

- ▶ **The code might be like :**

```
usa_capt = set('Washington')
nz_capt = set('Wellington')

print(usa_capt - nz_capt)
print(usa_capt.difference(nz_capt))
```

```
{'s', 'h', 'a'}
{'s', 'h', 'a'}
```

▶ Creating a set

- ▶ **The code might be like :**

```
usa_capt = set('Washington')
nz_capt = set('Wellington')

print(nz_capt - usa_capt)
print(nz_capt.difference(usa_capt))
```

```
{'l', 'e'}
{'l', 'e'}
```





▶ Creating a set

- ▶ **The code might be like :**

```
usa_capt = set('Washington')
nz_capt = set('Wellington')

print(nz_capt & usa_capt)
print(nz_capt.intersection(usa_capt))
```

```
{'i', 'o', 'g', 'n', 't', 'W'}
{'i', 'o', 'g', 'n', 't', 'W'}
```