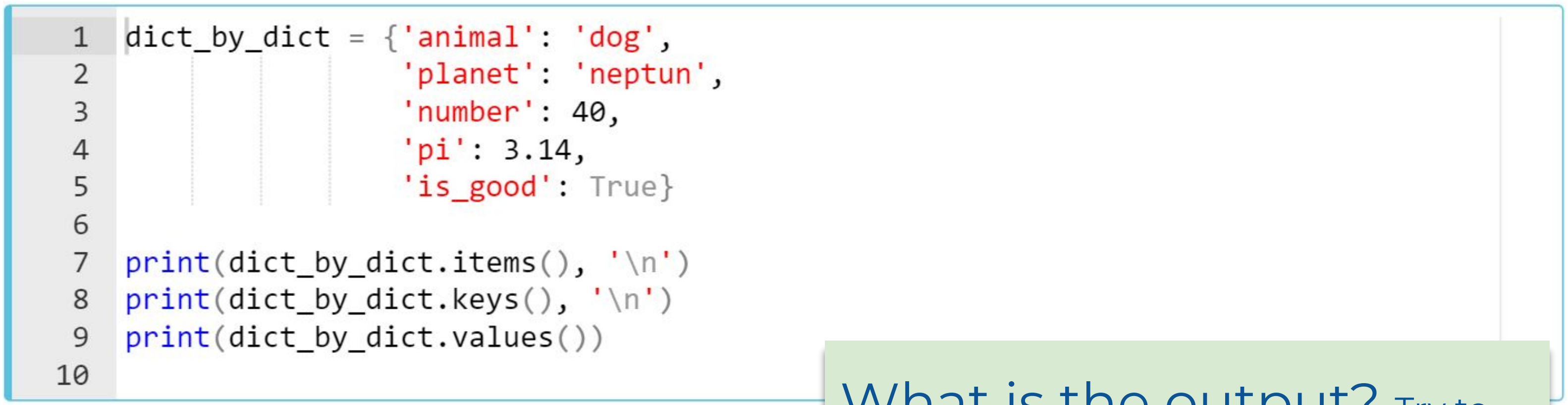
Main Operations with Dictionaries

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 :

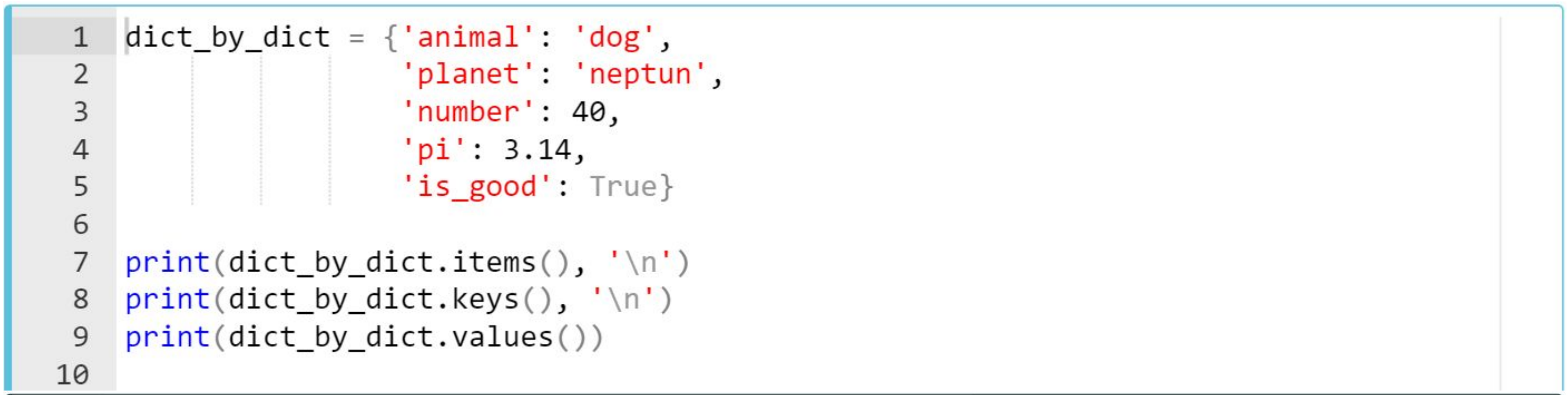


```python
dict_by_dict = {'animal': 'dog',
                'planet': 'neptun',
                'number': 40,
                'pi': 3.14,
                'is_good': True}

print(dict_by_dict.items(), '\n')
print(dict_by_dict.keys(), '\n')
print(dict_by_dict.values())
```

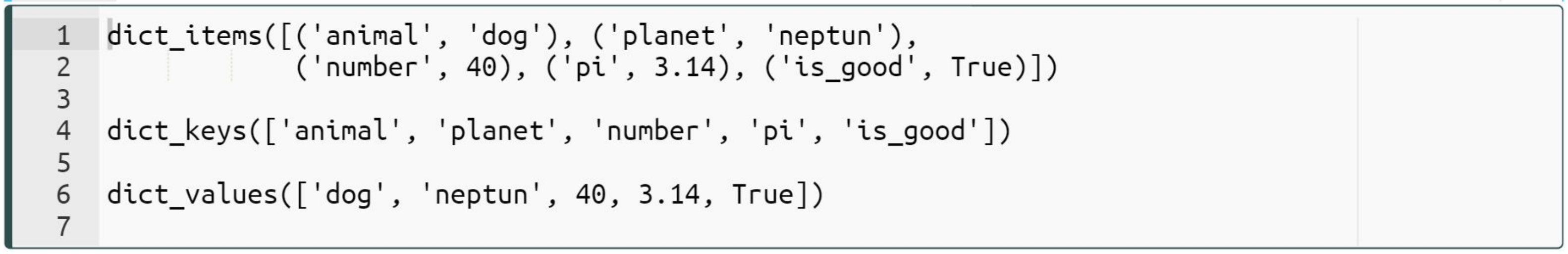
What is the output? Try to figure out in your mind...

Main Operations with `dicts` (review)

- Let's take a look at this example :



```python
dict_by_dict = {'animal': 'dog',
                'planet': 'neptun',
                'number': 40,
                'pi': 3.14,
                'is_good': True}

print(dict_by_dict.items(), '\n')
print(dict_by_dict.keys(), '\n')
print(dict_by_dict.values())
```



```
dict_items([('animal', 'dog'), ('planet', 'neptun'),
            ('number', 40), ('pi', 3.14), ('is_good', True)])

dict_keys(['animal', 'planet', 'number', 'pi', 'is_good'])

dict_values(['dog', 'neptun', 40, 3.14, True])
```

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 :

```python
dict_by_dict = {'animal': 'dog',
                'planet': 'neptun',
                'number': 40,
                'pi': 3.14,
                'is_good': True}

dict_by_dict.update({'is_bad': False})

print(dict_by_dict)
```

Main Operations with `dicts` (review)

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

```python
dict_by_dict = {'animal': 'dog',
                'planet': 'neptun',
                'number': 40,
                'pi': 3.14,
                'is_good': True}

dict_by_dict.update({'is_bad': False})

print(dict_by_dict)
```

```
{'animal': 'dog',
'planet': 'neptun',
'number': 40,
'pi': 3.14,
'is_good': True,
'is_bad': False}
```

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

CLARUSWAY© WAY TO REINVENT YOURSELF

Main Operations with `dicts`

- The code can be like :

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

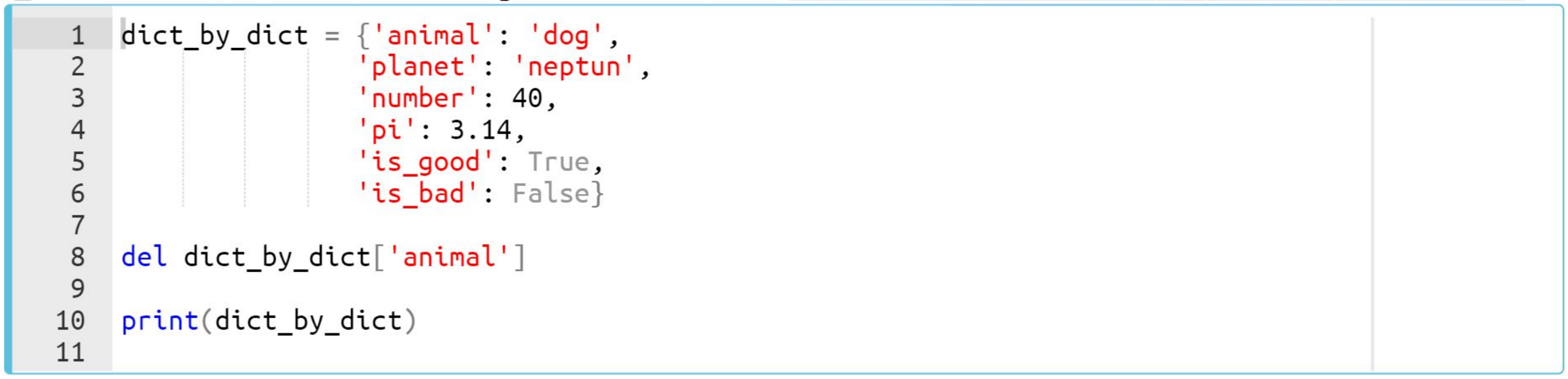
```
family = {'name1': 'Joseph',
          'name2': 'Bella',
          'name3': 'Aisha',
          'name4': 'Tom',
          'name5': 'Alfred'
         }
```

CLARUSWAY© WAY TO REINVENT YOURSELF

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']`

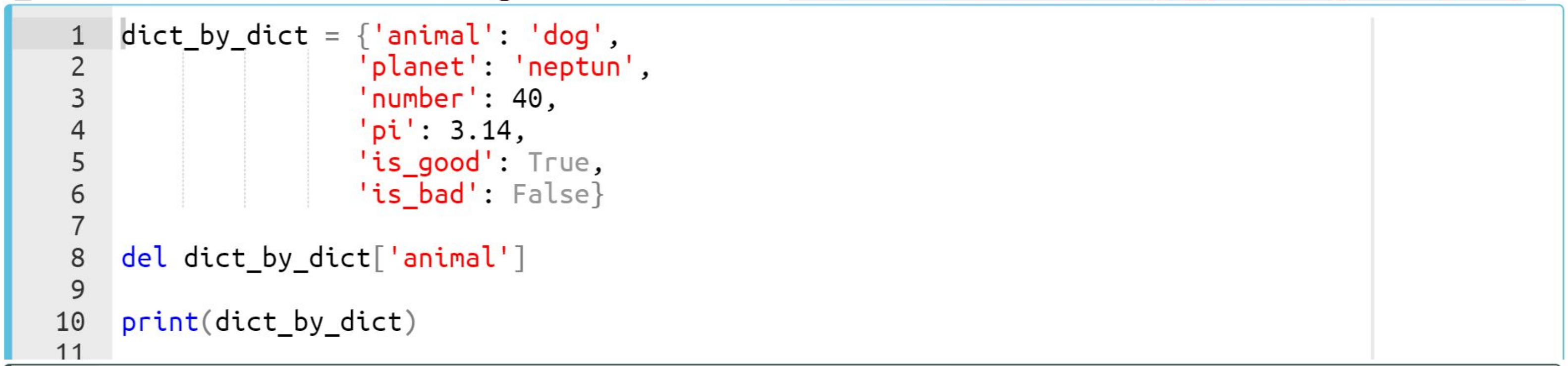


```python
dict_by_dict = {'animal': 'dog',
                'planet': 'neptun',
                'number': 40,
                'pi': 3.14,
                'is_good': True,
                'is_bad': False}

del dict_by_dict['animal']

print(dict_by_dict)
```

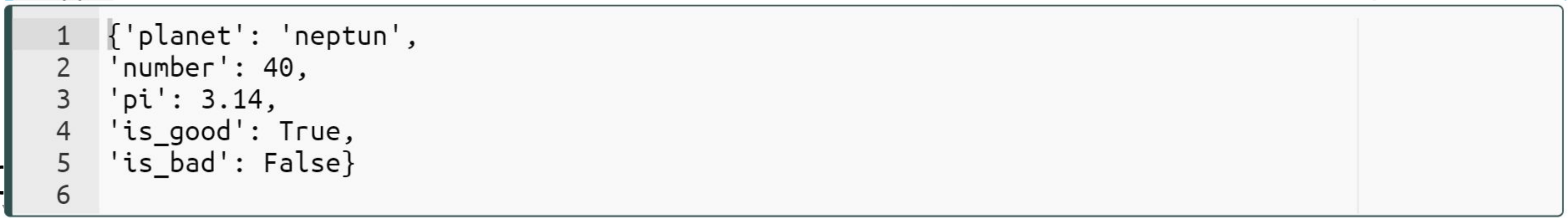
Main Operations with `dicts` (review)

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

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



```python
dict_by_dict = {'animal': 'dog',
                'planet': 'neptun',
                'number': 40,
                'pi': 3.14,
                'is_good': True,
                'is_bad': False}

del dict_by_dict['animal']

print(dict_by_dict)
```



```
{'planet': 'neptun',
'number': 40,
'pi': 3.14,
'is_good': True,
'is_bad': False}
```

Main Operations with `dicts`

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

CLARUSWAY©
WAY TO REINVENT YOURSELF

Main Operations with `dicts`

- The code can be like :

```
del family['name2']
del family['name3']

print(family)
```

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

CLARUSWAY©
WAY TO REINVENT YOURSELF

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'
         }
```

Students, write your response!

Main Operations with `dicts`

- The code can be like :

Option-1

```
del family['name2']
del family['name3']

print(family)
```

Option-2

```
del family['name2'], family['name3']

print(family)
```

```
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 `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`.

Main Operations with `dicts` (review)

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

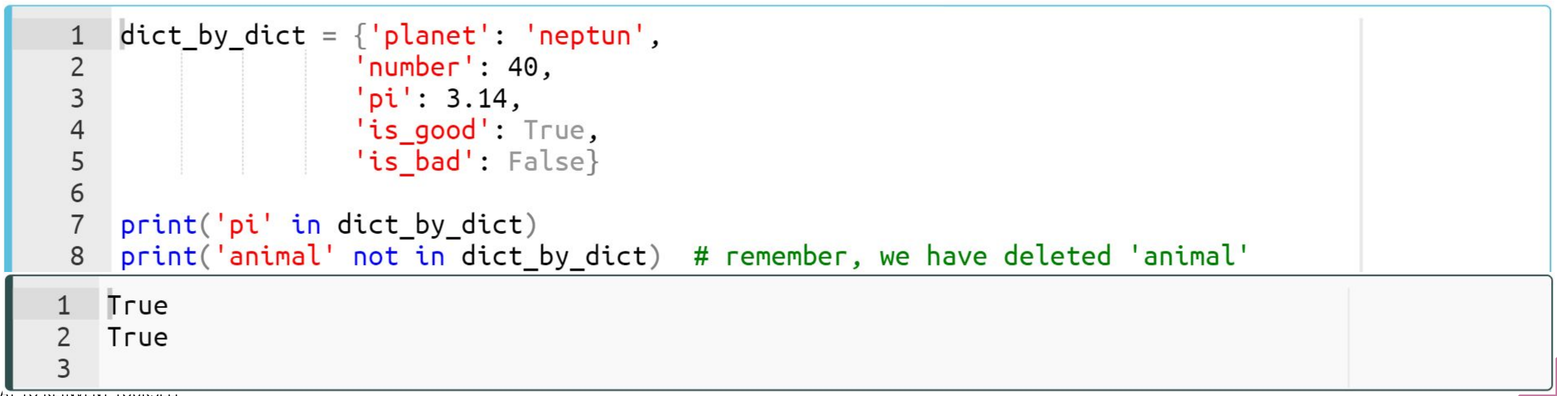
- When we use the `in` operator; if the `key` is in the dictionary, the result will be `True` otherwise `False`.
- When we use the `not in`; if the `key` is not in the dictionary, the result will be `True` otherwise `False`.

```python
dict_by_dict = {'planet': 'neptun',
                'number': 40,
                'pi': 3.14,
                'is_good': True,
                'is_bad': False}

print('pi' in dict_by_dict)
print('animal' not in dict_by_dict)  # remember, we have deleted 'animal'

```

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`.



```python
dict_by_dict = {'planet': 'neptun',
                'number': 40,
                'pi': 3.14,
                'is_good': True,
                'is_bad': False}

print('pi' in dict_by_dict)
print('animal' not in dict_by_dict)  # remember, we have deleted 'animal'
```

```
True
True
```

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 :

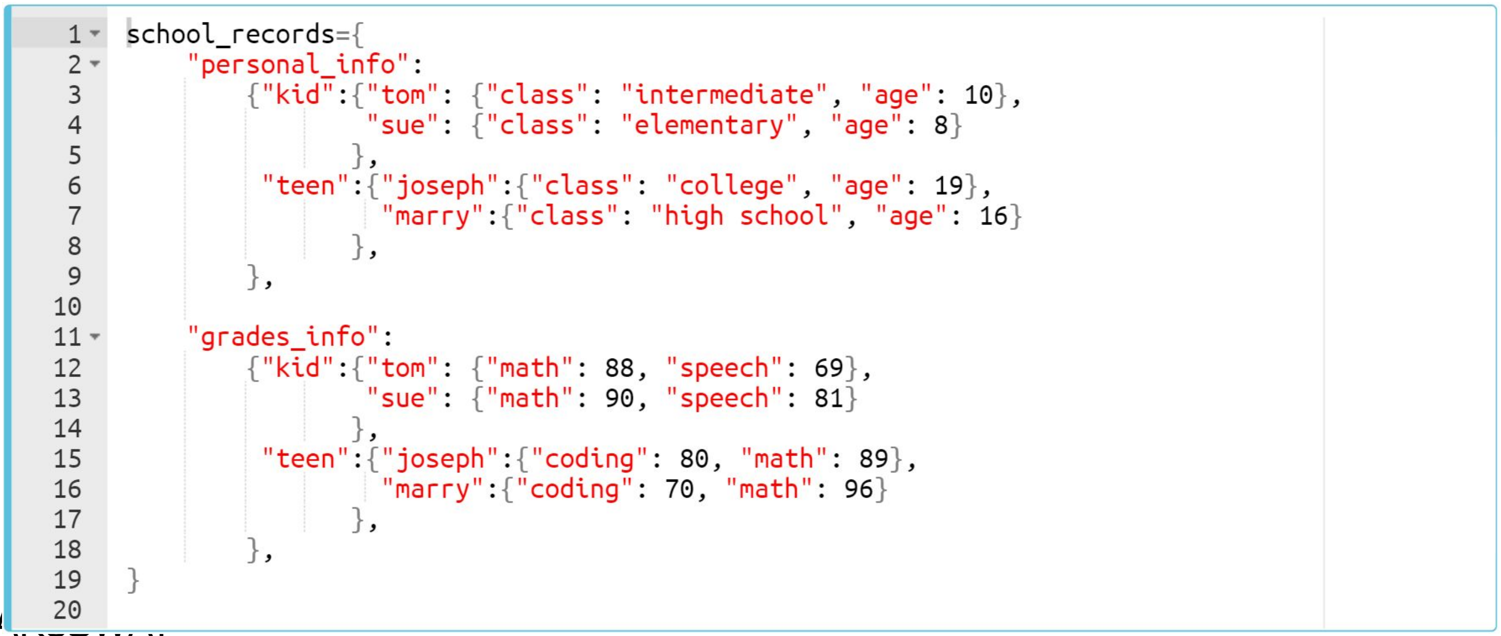
```
print('name3' in family)
```

```
False
```

Nested Dictionaries

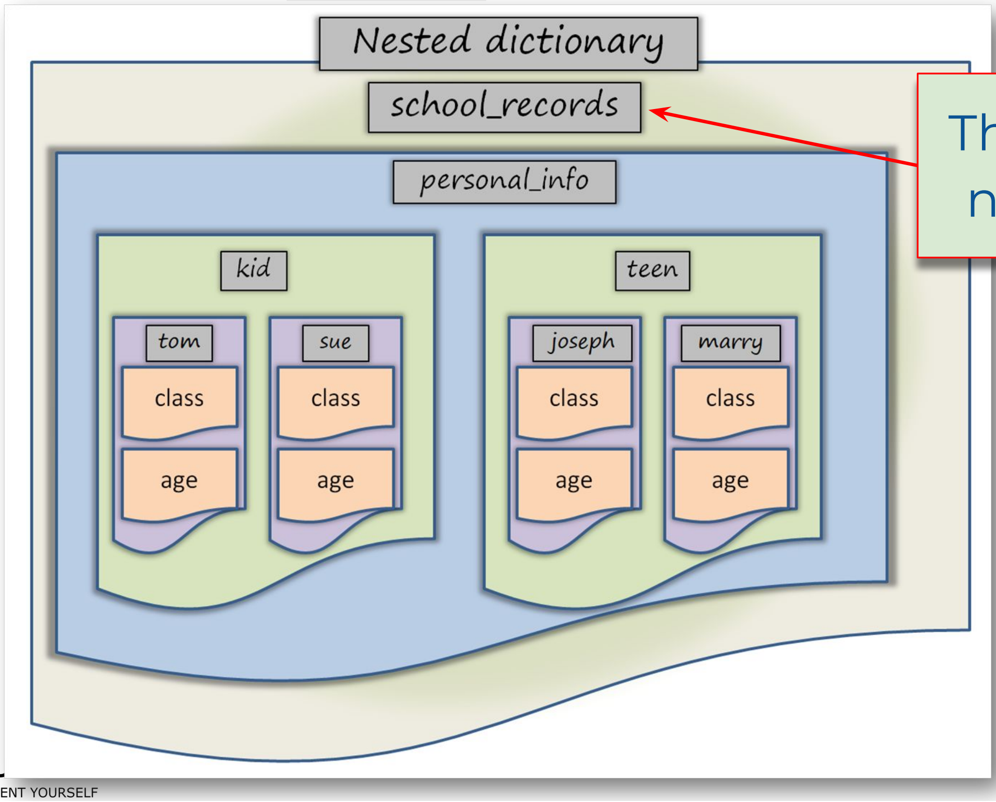
Nested `dicts` (review pre-class)

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



```
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}
               },
        },
}
```

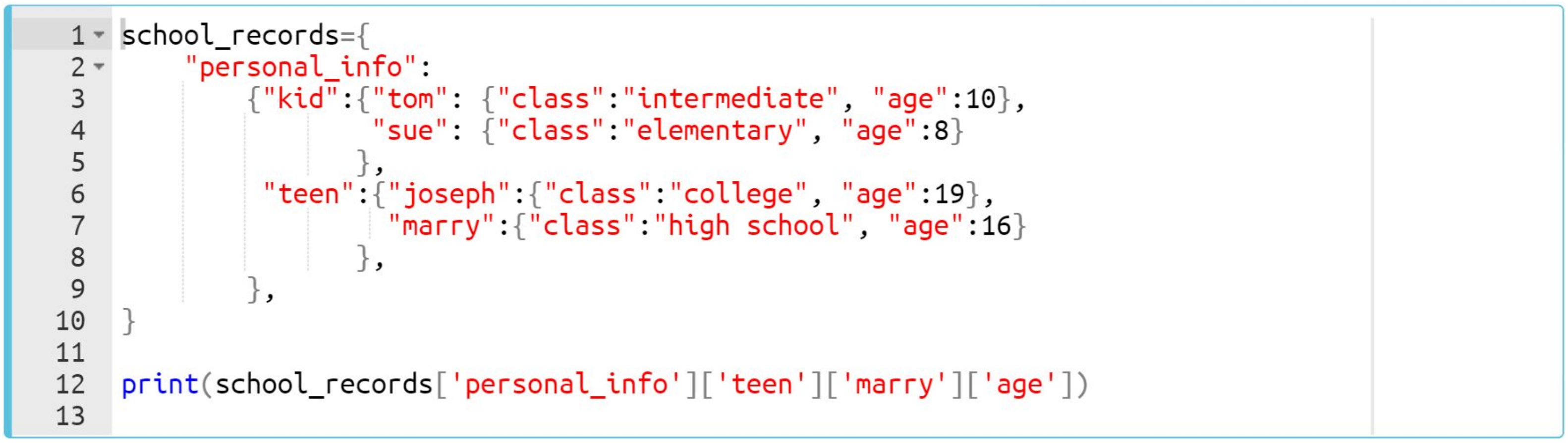
Nested `dicts` (review pre-class)



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.

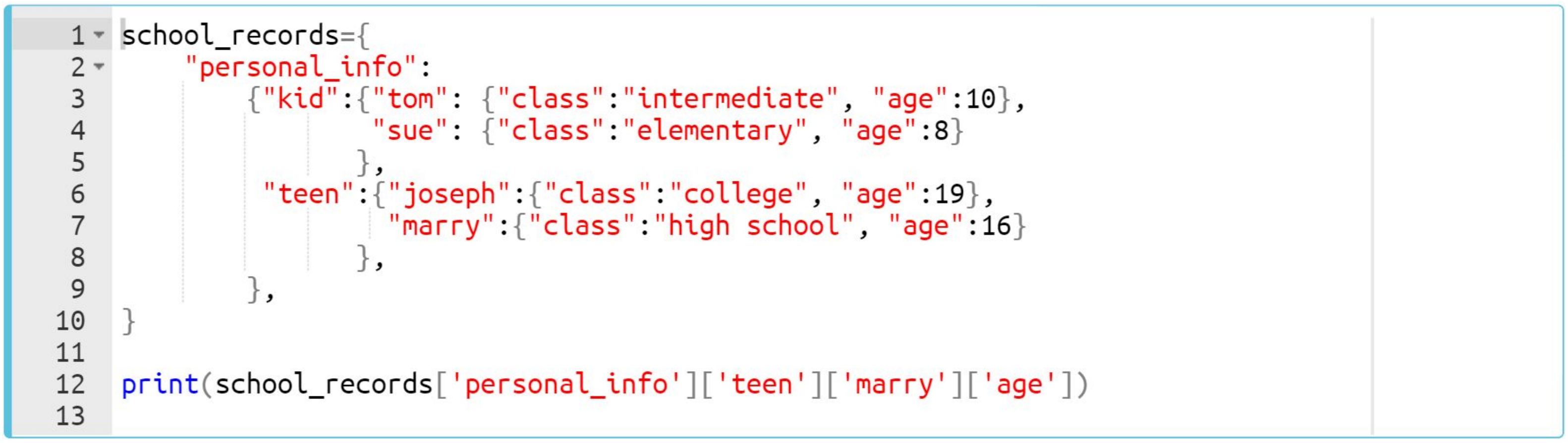


```python
school_records={
    "personal_info":
        {"kid":{"tom": {"class":"intermediate", "age":10},
                "sue": {"class":"elementary", "age":8}
               },
         "teen":{"joseph":{"class":"college", "age":19},
                 "marry":{"class":"high school", "age":16}
    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

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

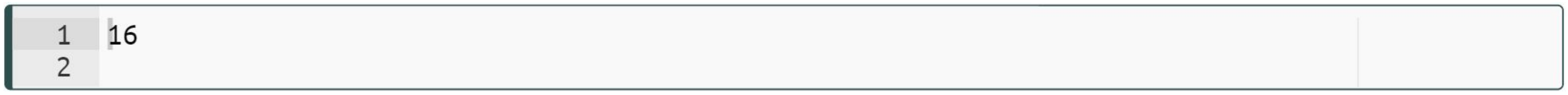
CLARUSWAY©
WAY TO REINVENT YOURSELF

Nested `dicts` (review pre-class)

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



```python
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'])
```

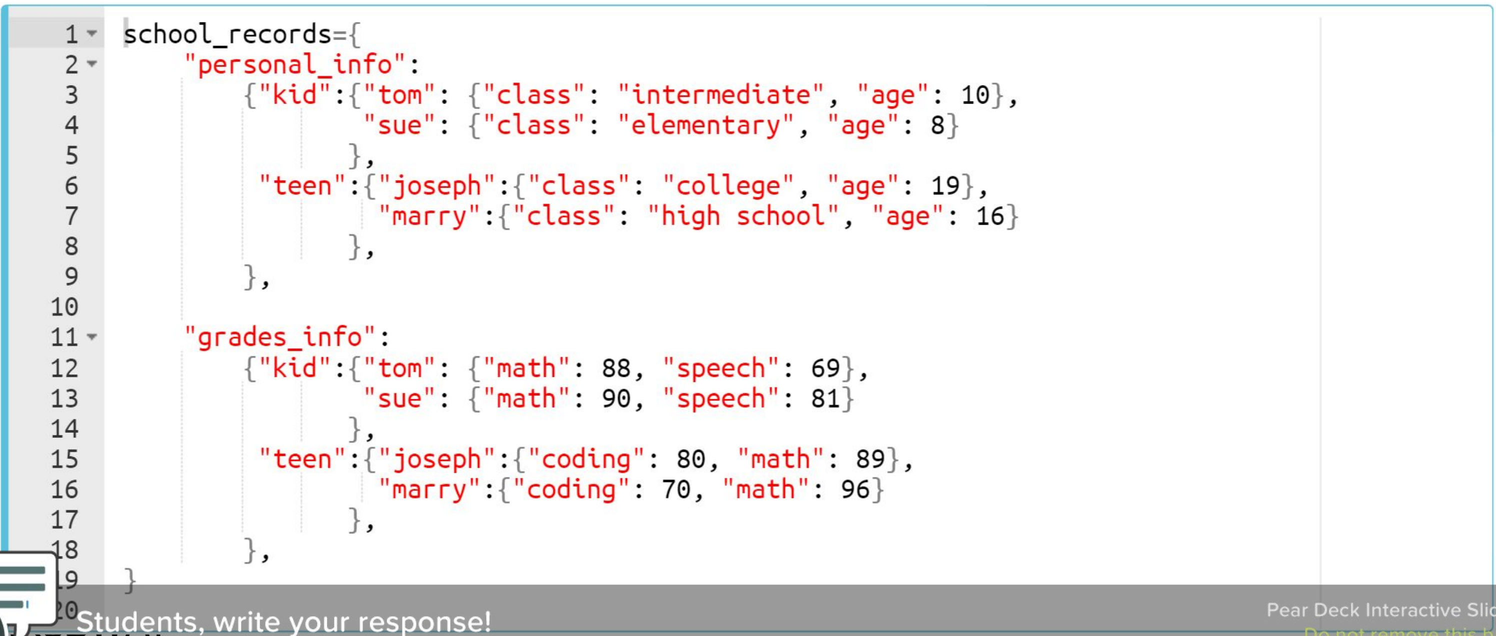


```
16
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

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`.



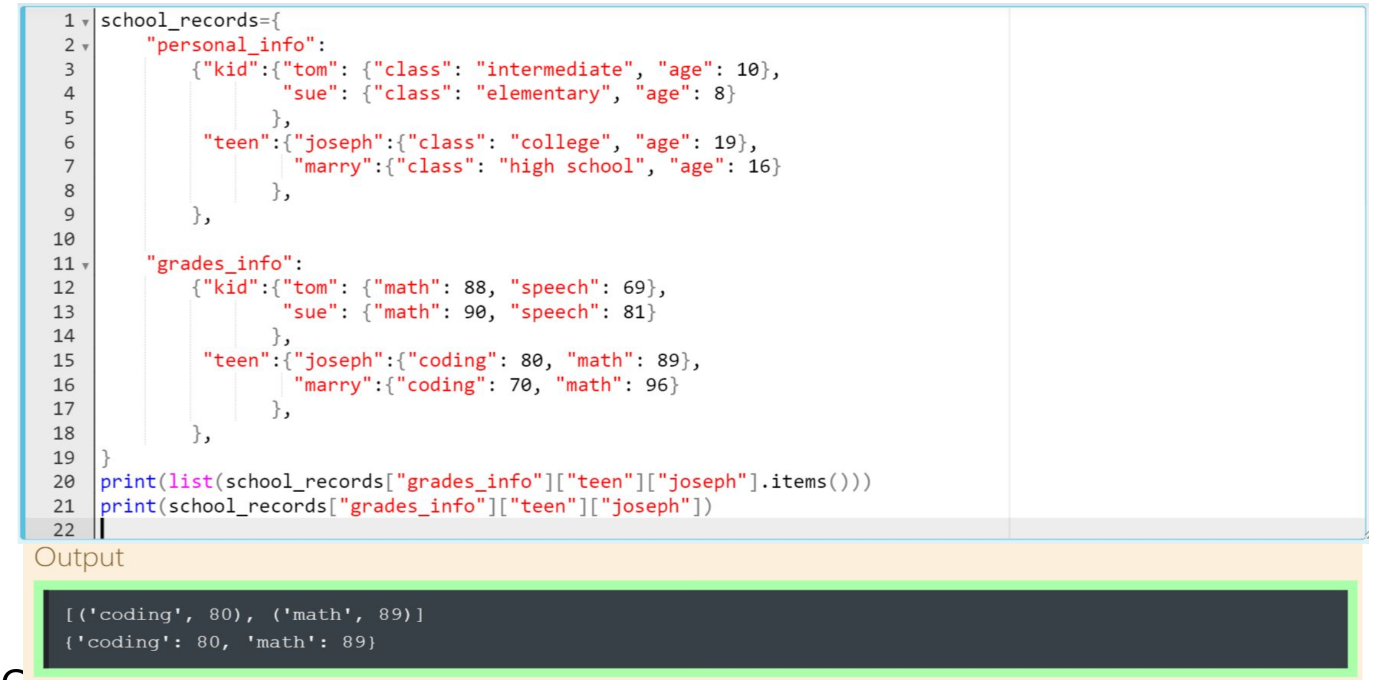
```python
school_records={
    "personal_info":
        {"kid":{"tom": {"class": "intermediate", "age": 10},
                "sue": {"class": "elementary", "age": 8}
              },
         "teen":{"joseph":{"class": "college", "age": 19},
                 "marry":{"class": "high school", "age": 16}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

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

Students, write your response!

Pear Deck Interactive Slide
Do not remove this bar

Nested `dicts`

- The code can be like :



```python
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}
              },
        },
}
print(list(school_records["grades_info"]["teen"]["joseph"].items()))
print(school_records["grades_info"]["teen"]["joseph"])
```

Output

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

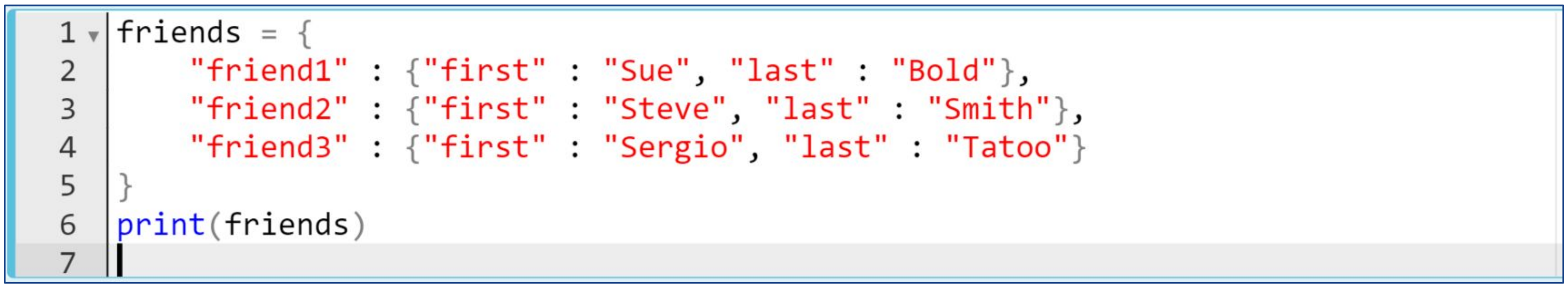
Nested `dicts`

- **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 :



```
friends = {
    "friend1" : {"first" : "Sue", "last" : "Bold"},
    "friend2" : {"first" : "Steve", "last" : "Smith"},
    "friend3" : {"first" : "Sergio", "last" : "Tatoo"}
}
print(friends)
```

Nested dicts

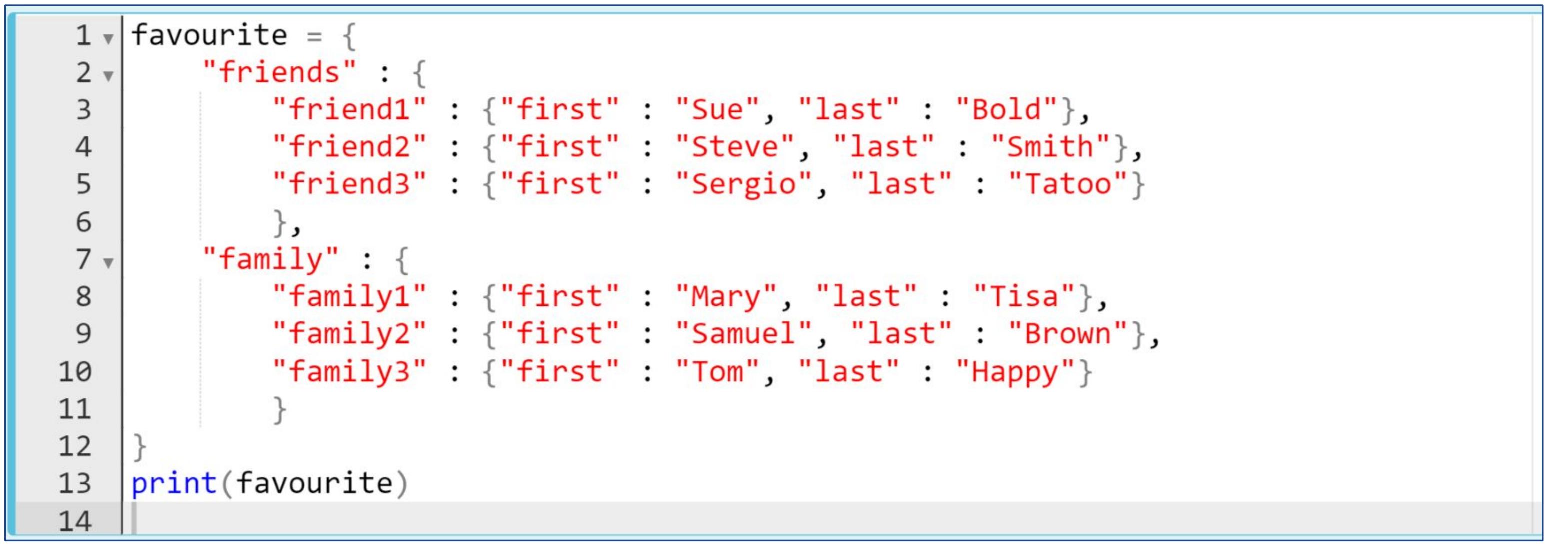
Create using curly braces 👉{}

- **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 :



```python
favourite = {
    "friends" : {
        "friend1" : {"first" : "Sue", "last" : "Bold"},
        "friend2" : {"first" : "Steve", "last" : "Smith"},
        "friend3" : {"first" : "Sergio", "last" : "Tatoo"}
        },
    "family" : {
        "family1" : {"first" : "Mary", "last" : "Tisa"},
        "family2" : {"first" : "Samuel", "last" : "Brown"},
        "family3" : {"first" : "Tom", "last" : "Happy"}
        }
}
print(favourite)

```

Sets

CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

CLARUSWAY©
WAY TO REINVENT YOURSELF

Definitions

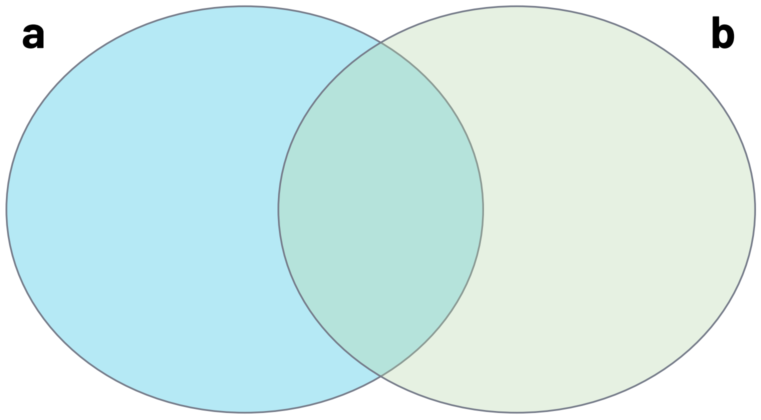
```python
fruit = {'Apple', 'Orange', 'Banana'}
```

```python
set()
```

CLARUSWAY©
WAY TO REINVENT YOURSELF

Definitions

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



CLARUSWAY©
WAY TO REINVENT YOURSELF

Creating a `set`

CLARUSWAY©
WAY TO REINVENT YOURSELF

Creating a `set`

- We have two basic ways to create a set.

- `{}`
- `set()`

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

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

CLARUSWAY©
WAY TO REINVENT YOURSELF

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)
```

⚠️ different order from the previous slide

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

Creating a set(review of pre-class)

- Here is an example of creating an empty set:

input :

```
empty_set = set()

print(type(empty_set))

```

output :

```
<class 'set'>

```

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.

CLARUSWAY©
WAY TO REINVENT YOURSELF

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

```python
flower_list = ['rose', 'violet', 'carnation', 'rose', 'orchid', 'rose', 'orchid']
flowerset = set(flower_list)
flowerlist = list(flowerset)

print(flowerset)
print(flowerlist)

```

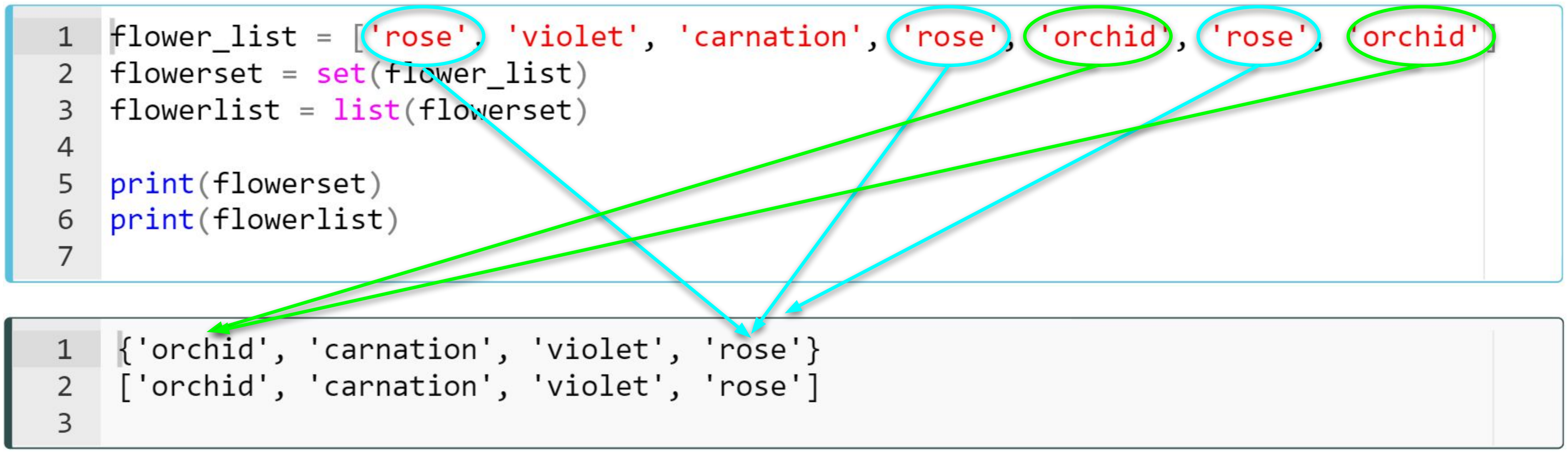
What is the output? Try to figure out in your mind...

Students, write your response!

CLARUSWAY©
REINVENT YOURSELF

Pear Deck Interactive Slide
Do not remove this bar

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


```
flower_list = ['rose', 'violet', 'carnation', 'rose', 'orchid', 'rose', 'orchid']
flowerset = set(flower_list)
flowerlist = list(flowerset)

print(flowerset)
print(flowerlist)
```

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

CLARUSWAY©
WAY TO REINVENT YOURSELF

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'}
```

Students choose an option

CLARUSWAY©
REINVENT YOURSELF

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)

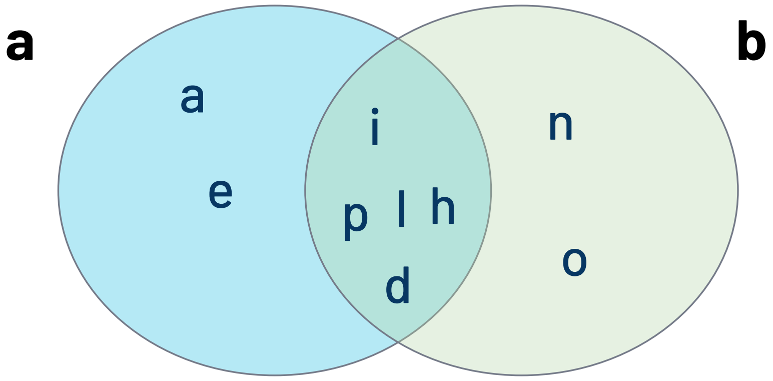
- The methods that can be used with `sets` :

- `.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')
```

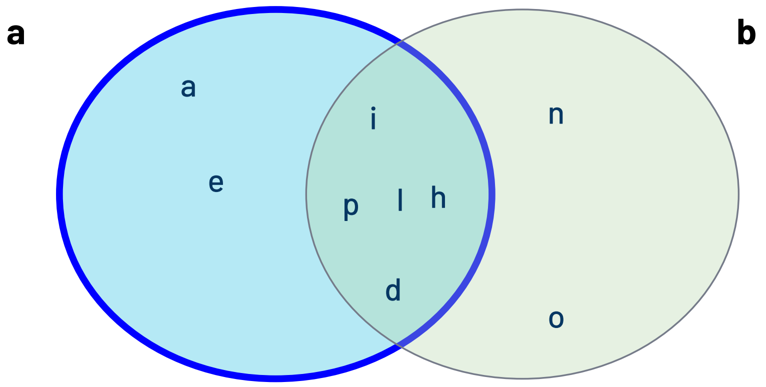


Main Operations with `sets`

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

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

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

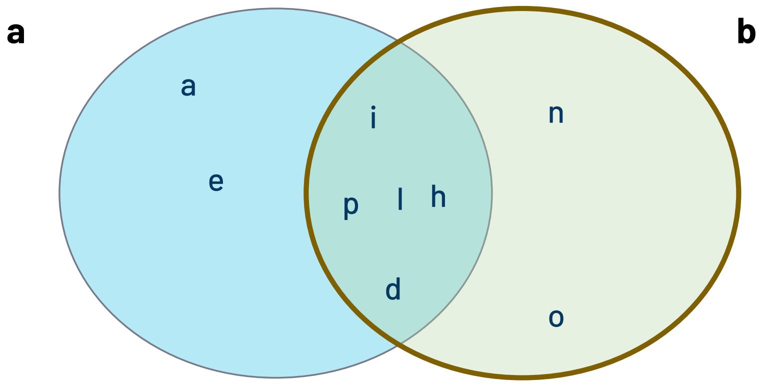


Main Operations with `sets`

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

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

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



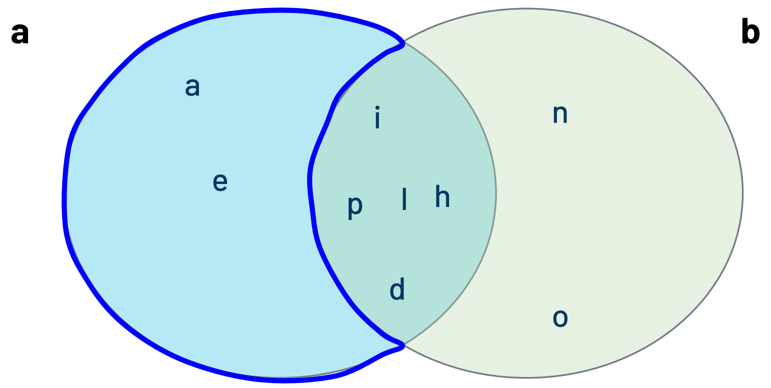
Main Operations with sets

- Basic set operations :

.difference(arg)

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

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



CLARUSWAY©
WAY TO REINVENT YOURSELF

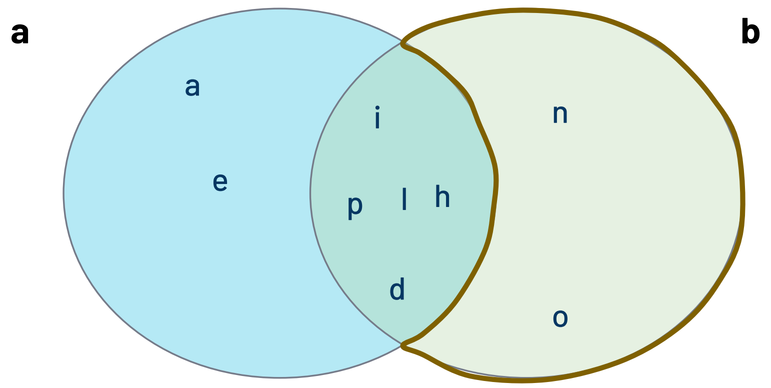
Main Operations with sets

- Basic set operations :

.difference(arg)

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

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



CLARUSWAY©
WAY TO REINVENT YOURSELF

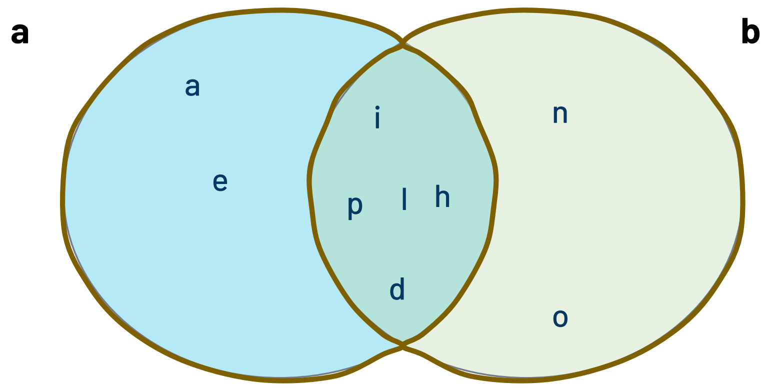
Main Operations with sets

- Basic set operations :

.union(arg)

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

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



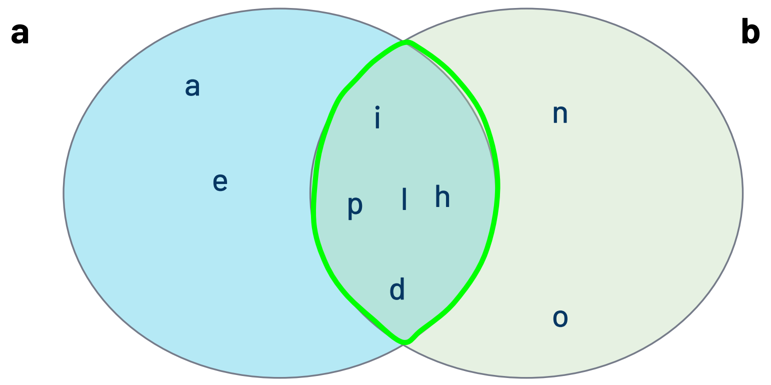
Main Operations with sets

- 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!

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'}
```