How can We Use a Tuple?

In one minute, write the usage of `tuples`..

Students, write your response!

How can We Use a `tuple`?

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

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

```

How can We Use a `tuple`?

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

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

```

```
0
2
4
-----------------------------------------------------------------------------
print(even_no[3]) : IndexError: tuple index out of range

```

How can We Use a `tuple`?

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

```python
city_list = ['Tokyo', 'Istanbul', 'Moskow', 'Dublin']

city_tuple = tuple(city_list)

city_tuple[0] = 'New York'  # you can't assign a value

```

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.

```python
city_list = ['Tokyo', 'Istanbul', 'Moskow', 'Dublin']

city_tuple = tuple(city_list)

city_tuple[0] = 'New York'  # you can't assign a value

```

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

```

Using Tuples

- **Task** :
  - Let's access, select and print the string 'six" from the following tuple. 👇

```
mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))

```

Using Tuples

- **The code should be like this : 👇**

```
mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))

str_six = mix_tuple[2][2][0]

print(str_six)

```

Using Tuples

- **Task** :
  - What is the output ? 👇

```
mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))

str_six = mix_tuple[2][1:3]

print(str_six, type(str_six), sep="\n")
```

Using Tuples

- **The output :** 👇

```python
mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))

str_six = mix_tuple[2][1:3]

print(str_six, type(str_six), sep="\n")

```

Try to figure out how the output can be like that?

Output

```
['two', ('six', 6)]
<class 'list'>
```

Using Tuples

- **Task** :
  - Access and print the **last** item and **its type** of the following tuple using **negative indexing** method : 👇

```
mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))

```

Using Tuples

- **The code should be like : 👇**

```python
mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))

last = mix_tuple[-1]

print(last, type(last), sep="\n")

```

Try to figure out how the output can be like that?

Output

```
(5, 'fair')
<class 'tuple'>
```

Using Tuples

- **Task** :
  - Let's access, select and print the "fair" of the following tuple. 👇 Use **two options** which consisting of *normal* and *negative* indexing methods.

```
mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))

```

Using Tuples

- **The code should be like : 👇**

```
mix_tuple = ("11", 11, [2, "two", ("six", 6)], (5, "fair"))

option_1 = mix_tuple[3][1]
option_2 = mix_tuple[-1][1]

print(option_1, option_2, sep = "\n")
```

Output

```
fair
fair
```

Refresh your mind with this interview question

Benefits of Immutability?

Try to write at least two things

Dictionaries

Table of Contents

Definitions

```
greengrocer = {'fruit' : 'Apple', 'vegetable' : 'Tomato'}
```

```
dict()
```

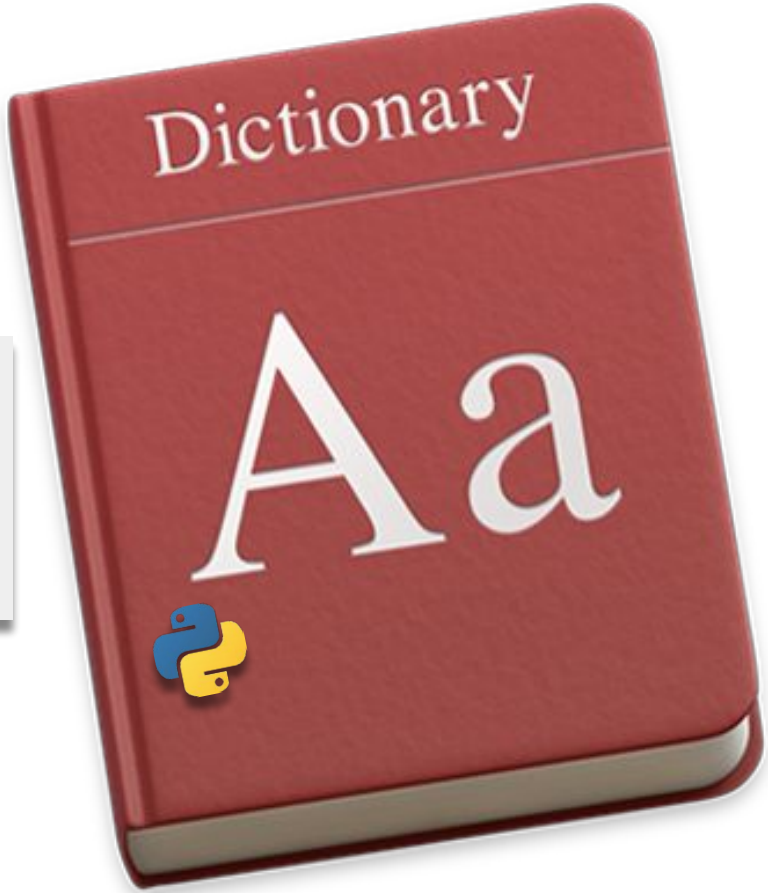
What did you learn from the pre-class content about dictionaries in Python?

Students, write your response!

Definitions

- Dictionaries

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



Creating a `dict`

Creating a `dict` (review)

- We have two basic ways to create a dictionary.

- `{}`
- `dict()`

Creating a `dict` (review pre-class)

- Here is an example of simple structure of a `dict`:

```
my_dict = {'key1': 'value1',
           'key2': 'value2',
           'key3': 'value3'
           }

```

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.

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

- `{}`
- `dict()`

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

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`(review pre-class)

- Accessing and assigning an item.

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

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

```

Creating a `dict` (review pre-class)

- Assigning a value to a key

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

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

```

```
Denver

```

Creating a `dict` (review pre-class)

- 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)

```

Creating a `dict` (review pre-class)

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

```python
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` (review pre-class)

Tips:

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

```
mix_values = {'animal': ('dog', 'cat'),  # tuple type
              'planet': ['Neptun', 'Saturn', 'Jupiter'],  # list type
              'number': 40,  # int type
              'pi': 3.14,  # float type
              'is_good': True}  # bool type

mix_keys = {22 : "integer",
            1.2 : "float",
            True : "boolean",
            "key" : "string"}

```

Creating a `dict`

- **Task** 👇
  - Let's create a `dict` (named `family`) which consists of names of 3 members of your family.
  - Each person should have only the first **names.**
  - For example;

    name1

    name2

    •

    •

Create using curly braces 👉 `{}`

Creating a `dict`

- The code can be like :

```
family = {'name1': 'Joseph',
          'name2': 'Bella',
          'name3': 'Aisha'
         }
```

Creating a `dict`

- **Task** 👇
  - Add a new family member name to the dictionary you created.

Creating a `dict`

- The code can be like :

```
family = {'name1': 'Joseph',
          'name2': 'Bella',
          'name3': 'Aisha'
         }
family['name4'] = 'Tom'
print(family)
```

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

Creating a `dict`

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

```
dict_by_dict = dict(animal='dog', planet='neptun', number=40, pi=3.14, is_good=True)

print(dict_by_dict)

```

Creating a `dict`

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

```python
dict_by_dict = dict(animal='dog', planet='neptun', number=40, pi=3.14, is_good=True)

print(dict_by_dict)

```

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

```

Creating a `dict`

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

```python
dict_by_dict = dict(animal='dog', planet='neptun', number=40, pi=3.14, is_good=True)

print(dict_by_dict)

```

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

```

⚠ **Avoid ! :**

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

Creating a `dict`

- **Task** 👇
  - Create the same `dict` using `dict()` function.

Creating a `dict`

- The code can be like :

```
family = dict(name1 = 'Joseph', name2 = 'Bella', name3 = 'Aisha',
name4 = 'Tom')

print(family)
```

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

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 :

```
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 :

```
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.

```
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.

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']`

```
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.

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'
         }
```

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`.

```
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 `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`.

```
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
```

Other Operations with `dicts`

- `clear()`; **Remove** all items from the dictionary.
- `pop(key[, default])`; If *key* is in the dictionary, **remove** it and **return its value**, else return *default*. If *default* is not given and *key* is not in the dictionary, a KeyError is raised.
- `popitem()`; **Remove** and **return a** `(key, value)` **pair** from the dictionary. Pairs are returned in **LIFO** order.
- `copy()`; Return a shallow **copy** of the dictionary.
- `get(key[, default])`; **Return the value** for *key* if *key* is in the dictionary, else *default*. If *default* is not given, it defaults to `None`, so that this method never raises a `KeyError`.
- `setdefault(key[, default])`; If *key* is in the dictionary, **return its value**. If not, **insert *key*** with a value of *default* and return *default*. *It* defaults to `None`.