Osnova

Lesson Overview

PYTHON ACADEMY / 3. DICTIONARIES & SETS / LESSON OVERVIEW

Long time, no see! Welcome back to Python Academy.

So far you know:

REVIEW EXERCISES

- how to use conditions
- how to operate with basics data types.

It´s time for learn something new. In lesson 3, we will cover:

- **dictionaries**,
- **sets**.

Dictionary is a new data type and is pretty much like phonebook, where you add various values (phone number) to users (contact name). When you select specific user (contact name), dictionary will give you information with added value (phone number). Easy peasy.

And sets? Sets are also a new data type and is similar to sets you may remember from math at primary school. It is simply a collection of unique objets, but more on that in the lesson ;)

100% z Lekce 3

Osnova

00:31

REVIEW EXERCISES

Find bugs

PYTHON ACADEMY / 3. DICTIONARIES & SETS / REVIEW EXERCISES / FIND BUGS

100% z Lekce 3

Find a [illegible] following code:

Osnova

```
some_str = input('Hello, please enter anything you want: ")

if isupper(some_str):
    print('All letters in your input are uppercase')
else if islower(some_str):
    print('All letters in your input are lowercase')
else
    print('There are both uppercase and lowercase letters')
```

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

The best way to find a bug in a program is to run that program:

100% z Lekce 3

```
  File "buggy_review.py", line 1
    Osnova
    input('Hello, please enter anything you want")
                                                  ^
SyntaxError: EOL while scanning string literal
```

- Python has just showed me, that there is a problem in the input function call - specifically at the end I have double quotes `"` instead of single quotes that should match the beginning of the string.
- Let's fix that and run the program again

```
~/PythonBeginner/Lesson2 $ python buggy_review.py
  File "buggy_review.py", line 5
    else if islower(some_str):
          ^
SyntaxError: invalid syntax
```

- Now there is a problem - SyntaxError - on line 5
- Python does not recognize the structure `else if` - the correct code would be `elif`
- Let's fix that and run the program again

```
~/PythonBeginner/Lesson2 $ python buggy_review.py
  File "buggy_review.py", line 7
    else
       ^
SyntaxError: invalid syntax
```

- On line 7 we forgot to include `:` after the else keyword
- Let's fix that and run the program again

```
~/PythonBeginner/Lesson2 $ python buggy_review.py
Hello, please enter anything you want: abc
Traceback (most recent call last):
  File "buggy_review.py", line 3, in <module>
    if isupper(some_str):
NameError: name 'isupper' is not defined
```

- we fixed all SyntaxError first and now we have to deal with so called runtime errors - in this case `isupper` should be a method

- The problem we will have with `islower` call in the `elif` case

Osnova

- lets fix those two calls as follows:

```
some_str = input('Hello, please enter anything you want')

if some_str.isupper(some_str):
    print('All letters in your input are uppercase')
elif some_str.islower(some_str):
    print('All letters in your input are lowercase')
else:
    print('There are both uppercase and lowercase letters')
```

- Running the program we get yet another error:

```
~/PythonBeginner/Lesson2 $ python buggy_review.py
Hello, please enter anything you want: abc
Traceback (most recent call last):
  File "buggy_review.py", line 3, in <module>
    if some_str.isupper(some_str):
TypeError: isupper() takes no arguments (1 given)
```

- Neither `isupper()` nor `islower()` method takes any inputs into parentheses - we should erase them:

```
some_str = input('Hello, please enter anything you want: ')

if some_str.isupper():
    print('All letters in your input are uppercase')
elif some_str.islower():
    print('All letters in your input are lowercase')
else:
    print('There are both uppercase and lowercase letters')
```

FINALLY, the program runs correctly:

```
~/PythonBeginner/Lesson2 $ python buggy_review.py
Hello, please enter anything you want: abc
All letters in your input are lowercase
```

100% z Lekce 3

Osnova

```
if some_str.isupper():
    print('All letters in your input are uppercase')
elif some_str.islower():
    print('All letters in your input are lowercase')
else:
    print('There are both uppercase and lowercase letters')
```

Splitting Numbers

REVIEW EXERCISES

Create a script `split_nums.py` that will:

- ask the user for a number
- split the given number in halves (e.g. 123456 -> split to 123 and 456, 12345 -> 12 and 345)
- convert both halves into an integer
- if both halves are an even integer, print: `'Success'` - e.g. 12 and 34
- if only the first part is even, print: `'First'` - e.g. 12 and 345
- if the second part is even, print: `'Second'` - e.g. 123 and 456
- if neither of the numbers is even print: `'Neither'` - 123 and 455
- if nothing has been entered (the user just hit Enter), print: `'No input provided'`

Example of running the script:

```
/Users/PythonBeginner/Lesson1$ python split_nums.py
Please, give me a number: 35
Neither
```

```
/Users/PythonBeginner/Lesson1$ python split_nums.py
Please, give me a number:
```

100% z Lekce 3

Online Python Editor

Osnova

You can create the script in your computer or here in our editor.

```
1
```

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

100% z Lekce 3

2. So, if we have an input, we can work further with it

Osnova

3. We want to split the input string into 2 halves. To find the **midpoint**, we apply the **floor division**
4. After we have extracted the first and the second half of the numeric string, we can convert them into integers
5. Lastly, we want to compose a conditional statement

```
num = input('Please, give me a number: ')

# 1
if num == '':
    print('No input provided')
# 2
else:
    # 3
    mid_point = len(num) // 2
    # 4
    first_half = int(num[:mid_point])
    second_half = int(num[mid_point:])
    # 5
    if first_half % 2 == 0 and second_half % 2 == 0:
        print('Success')
    elif first_half % 2 == 0:
        print('First')
    elif second_half % 2 == 0:
        print('Second')
    else:
        print('Neither')
```

Password check 1

PYTHON ACADEMY / 3. DICTIONARIES & SETS / REVIEW EXERCISES / PASSWORD CHECK 1

100% z Lekce 3

- [illegible] for the secret password

Osnova

- If the password given, starts with any of the lowercase 'a','e','f','q','z', print to the terminal `'Welcome!'`
- otherwise print `'The input does not match'`

Example of running the script:

```
/Users/PythonBeginner/Lesson1$ python check_start.py
Please enter the password: abcd
Welcome!
```

```
/Users/PythonBeginner/Lesson1$ python check_start.py
Please enter the password: black hand
The input does not match
```

REVIEW EXERCISES

Online Python Editor

```
1
```

100% z Lekce 3

Osnova

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

We can see from the task description, that there are two possible outcomes from the program:

REVIEW EXERCISES

1. print `'Welcome!'`
2. **otherwise** print `'The input does not match'`

That tells us that we have to set up a conditions made of two branches - `if - else` .

To check, whether a string starts with a given letter, we have to extract this letter first using indexing. The first letter encounters itself at index 0.

To check, whether the first letter is one of the letters 'a','e','f','q','z', we just need to use **membership testing** applied to a string of those letters `if password[0] in 'aefqz` .

```
password = input('Please enter the password: ')
if password[0] in 'aefqz':
    print('Welcome!')
else:
    print('The input does not match')
```

100% z Lekce 3

Osnova

REVIEW EXERCISES

Create a dictionary

PYTHON ACADEMY / 3. DICTIONARIES & SETS / ONSITE PROJECT / CREATE A DICTIONARY

In the few following tasks, we will be working with the script `films.py` .

Please create a new script called `films.py` . Inside the script create a dictionary called `film` with the following mapping:

```
name = 'Shawshank Redemption'
rating = 87
year = 1994
director = 'Frank Darabont'
```

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

100% z Lekce 3

Click to see our solution

Osnova

There are multiple ways we can create a dict:

1. Starting from an empty dict
   - assigning value to a key:

```
film = {}
film['name'] = 'Shawshank Redemption'
film['rating'] = 87
film['year'] = 1994
film['director'] = 'Frank Darabont'
```

   - using `dict.update()` method:

```
film = {}
film.update(name = 'Shawshank Redemption')
film.update(rating = 87)
film.update(year = 1994)
film.update(director = 'Frank Darabont')
```

2. Listing key-value pairs immediately at the moment of creation

```
film = {'name': 'Shawshank Redemption', 'rating': 87, 'year': 1994,
'director': 'Frank Darabont' }
```

Add a new category

PYTHON ACADEMY / 3. DICTIONARIES & SETS / ONSITE PROJECT / ADD A NEW CATEGORY

Inside your script `films.py`, write the code, that will add 2 new categories to the dictionary `film`.

100% z Lekce 3

- The first new category should called `'starring'` and we should associate a with it a list containing `'Tim Robbins'`, `'Morgan Freeman'`.

Osnova

- the other category should be called `'budget'` and the value associated with it should be 200

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

We can update a dictionary in the 2 following ways:

REVIEW EXERCISES

1. assigning value to a key:

```
film['starring'] = ['Tim Robbins', 'Morgan Freeman']
film['budget'] = 200
```

2. using `dict.update()` method.
   - note that the category name is not enclosed in quotes. Python will do that for us.

```
film.update(starring = ['Tim Robbins', 'Morgan Freeman'])
film.update(budget = 200)
```

Dictionary `film` should like this now:

```
>>> film
{'name': 'Shawshank Redemption', 'starring': ['Tim Robbins', 'Morgan Freeman'], 'year': 1994, 'budget': 200, 'director': 'Frank Darabont', 'rating': 87}
```

100% z Lekce 3

In the previous step, we have made mistake by adding a new category `'budget'` and therefore we should remove it. Please remove the whole key - value pair under the `'budget'` key.

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

Again, there are multiple ways, how a key:value pair can be erased from the dictionary. If we know, which key:value pair we want to remove, we can use:

1. the `del` keyword infront of the dictionary retrieving the given key:

```
del film['budget']
```

2. `dict.pop(key)` method, which moreover returns the key of the removed key : value pair

```
film.pop('budget')
```

3. `dict.popitem()` method, which moreover returns the whole key : value pair being removed

```
film.popitem()
```

The film dictionary should not contain the `'budget'` category anymore:

```
{'name': 'Shawshank Redemption', 'starring': ['Tim Robbins', 'Morgan Freeman'], 'year': 1994, 'director': 'Frank Darabont', 'rating': 87}
```

Osnova ithin another dict

Do the following:

1. Create a new empty dictionary called `films` .
2. Add the dictionary you have stored inside the variable `film` to the newly created `films` dictionary under the key 'DRAMA'.
3. Print the content of the `films` dictionary:

Example of program execution:

```
~/PythonBeginner/Lesson2 $ python films.py
{'DRAMA': {'year': 1994, 'starring': ['Tim Robbins', 'Morgan Freeman'],
'rating': 87, 'director': 'Frank Darabont', 'name': 'Shawshank
Redemption'}}
```

Note that the order in which the categories are printed does not have to be necessarily the same as the one depicted above.

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

Here we are doing nothing new compared to the previous tasks concerning dict manipulation.

We need to:

1. Create a ditionary:

100% z Lekce 3

6. Create new key : value pair inside it. The new key value pair will be `'DRAMA' : film` :

Osnova

```
films['DRAMA'] = film
```

OR

```
films.update(DRAMA =film)
```

Our dictionary should look like this:

```
{'DRAMA': {'name': 'Shawshank Redemption', 'starring': ['Tim Robbins', 'Morgan Freeman'], 'year': 1994, 'director': 'Frank Darabont', 'rating': 87}}
```

REVIEW EXERCISES

Dictionary querying

PYTHON ACADEMY / 3. DICTIONARIES & SETS / ONSITE PROJECT / DICTIONARY QUERYING

Now we have a very (very) small film database. We can now offer a user to ask for information regarding the films we store in there.

The user should be able to request information according to film genre. Try to write the code that will make the program as demonstrated below.

Example of running the program:

```
~/PythonBeginner/Lesson2 $ python films.py
We can currently offer:
['DRAMA']
What genre are you interested in? DRAMA
HERE IT GOES:
{'year': 1994, 'starring': ['Tim Robbins', 'Morgan Freeman'], 'rating':
```

100% z Lekce 3

Code Solution

Osnova

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

- here we have to be a bit creative with our current knowledge base
- what will happen, if we pass dictionary `films` into a `list()` function? We get the list of keys in the top level of the dictionary `films`
- we can print the list of keys to the terminal and then prompt the user to choose one of those (lines 2 - 3)
- in order to create one blank line btw. the prompt to choose the genre and the `'HERE IT GOES'`, we can use `print()` function with no inputs
- once we have the input from the user, we can use it as a key to query the data, user wants to see and print them to the screen

REVIEW EXERCISES

```
1 print('We can currently offer:')
2 print(list(films))
3 genre = input('What genre are you interested in? ')
4 print()
5 print('HERE IT GOES')
6 print(films[genre])
```

Clearing the dictionary

PYTHON ACADEMY / 3. DICTIONARIES & SETS / ONSITE PROJECT / CLEARING THE DICTIONARY

Our last task is to write a command, that will empty the whole dictionary `films` and print the result to the terminal after printing the message `DATABASE HAS BEEN ERASED:` .

100% z Lekce 3

```
/P[illegible] Lesson2 $ python films.py
[illegible] offer:
['DRAMA']
What genre are you interested in? DRAMA
HERE IT GOES:
{'year': 1994, 'starring': ['Tim Robbins', 'Morgan Freeman'], 'rating':
87, 'director': 'Frank Darabont', 'name': 'Shawshank Redemption'}
DATABASE HAS BEEN ERASED:
{}
```


Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

The simplest way to erase a whole dictionary is to use `dict.clear()` method:

```
1 films.clear()
2 print('DATABASE HAS BEEN ERASED:')
3 print(films)
```

Dictionary Code Solutions

PYTHON ACADEMY / 3. DICTIONARIES & SETS / ONSITE PROJECT / DICTIONARY CODE SOLUTIONS

So this is it for playing with dictionary. Check out the whole solution, before we move to sets.

Click to see our solution

Osnova

```
film['title'] = 'Shawshank Redemption'
film['rating'] = 87
film['year'] = 1994
film['director'] = 'Frank Darabont'

film.update(starring = ['Tim Robbins', 'Morgan Freeman'])
film.update(budget = 200)

del film['budget']

films = {}
films.update(DRAMA=film)

print('We can currently offer:')
print(list(films))
genre = input('What genre are you interested in? ')
print()
print('HERE IT GOES')
print(films[genre])

films.clear()
print('DATABASE HAS BEEN ERASED:')
print(films)
```

Items in common

PYTHON ACADEMY / 3. DICTIONARIES & SETS / ONSITE PROJECT / ITEMS IN COMMON

Now, let's play with `Sets` for a bit. In the following exercises, we will be working with the following 2 strings:

- str1 = 'New York'

100% z Lekce 3

Write a [illegible] script (you can call it `common.py`) that will print all the letters that these two strings have.

Osnova

Example of running the script:

```
~/PythonBeginner/Lesson2 $ python common.py
['Y', 'e', 'k', 'r', 'o']
```

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

REVIEW EXERCISES

- first we have to store both strings in their variables `str1` and `str2`
- subsequently we need to extract letters that exist in both these strings
- for that purpose we can use set operation of **intersection** `&`
- in order we can apply intersection to letters, we need to convert both strings into set using `set()` function
- we do these two steps on one line (no. 3) and we store the result of the intersection inside the variable `in_common`
- we print the variable content of `in_common` converted into a list using `list()` function (line 4)

```
1 str1 = 'New York'
2 str2 = 'Yorkshire'
3 in_common = set(str1) & set(str2)
4 print(list(in_common))
```

100% z Lekce 3

In the following exercise, we will be working with the following 2 strings:

- str1 = 'New York'
- str2 = 'Yorkshire'

Write a short script (we can call it `unique.py`) that will print:

- a list of letters that are unique to str1
- and then list of letters unique to str2

Example of running the script:

```
~/PythonBeginner/Lesson2 $ python unique.py
Unique to str1: [' ', 'w', 'N']
Unique to str2: ['h', 's', 'i']
```

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

- first we have to store both strings in their variables `str1` and `str2`
- subsequently we need to extract letters that exist in `str1` but not in `str2` and vice versa
- for that purpose we can use set operation of **difference** `-`
- in order we can apply difference to letters, we need to convert both strings into sets using `set()` function
- we first perform the difference among `str1` and `str2`
- then we request the difference among `str2` and `str1`

- [illegible] we convert the content of `unique_str1` and `unique_str2` into list
  Osnova
- also, in order we do not have to convert list to the string, we can separate the acompanying string from the list by a comma

```
str1 = 'New York'
str2 = 'Yorkshire'
# Difference
unique_str1 = set(str1) - set(str2)
unique_str2 = set(str2) - set(str1)
print('Unique to str1:', list(unique_str1))
print('Unique to str1:', list(unique_str2))
```

REVIEW EXERCISES

Symmetric difference

PYTHON ACADEMY / 3. DICTIONARIES & SETS / ONSITE PROJECT / SYMMETRIC DIFFERENCE

In the following exercise, we will be working with the following 2 strings:

- str1 = 'New York'
- str2 = 'Yorkshire'

Write a short script (we can call it `unique_together.py`) that will print letters that are in str1 or str2 but not in both

Example of running the script:

```
~/PythonBeginner/Lesson2 $ python unique_together.py
[' ', 'w', 's', 'i', 'h', 'N']
```

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

100% z Lekce 3

Click to see our solution

Osnova

- in other words we want to add together the difference among str1 and str2 and vice versa
- we can do this using **symmetric difference** operation - `^`

```
str1 = 'New York'
str2 = 'Yorkshire'
symmetric_diff = set(str1) ^ set(str2)
print(list(symmetric_diff))
```

REVIEW EXERCISES

Union

PYTHON ACADEMY / 3. DICTIONARIES & SETS / ONSITE PROJECT / UNION

In the following exercise, we will be working with the following 2 strings:

- str1 = 'New York'
- str2 = 'Yorkshire'

Write a short script (we can call it `both.py`) that will print the list of unique letters (symbols), that can be found in str1 or str2.

Example of running the script:

```
~/PythonBeginner/Lesson2 $ python both.py
[' ', 'k', 'e', 'r', 'w', 's', 'i', 'o', 'h', 'Y', 'N']
```

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

100% z Lekce 3

Click to see our solution

Osnova

- first we have to store both strings in their variables `str1` and `str2`
- subsequently we need to extract letters that exist in `str1` or `str2`
- for that purpose we can use set operation of **union** `|`
- let's convert our strings into sets and apply union operation to them, let's store the result in a variable `united`
- at the end we just print out `united` converted into a list

```
str1 = 'New York'
str2 = 'Yorkshire'
united = set(str1) | set(str2)
print(list(united))
```

REVIEW EXERCISES

School subject attendance

PYTHON ACADEMY / 3. DICTIONARIES & SETS / ONSITE PROJECT / SCHOOL SUBJECT ATTENDANCE

We have information about the students inscribed into 5 classes in our school. We would like to find out which students attend all the classes. Create a Python script called `class_stats.py`

Here we have the information:

```
classes = {'Biology' :
['Adam','Chelsea','Marcus','Oliver','Alex','Sandra','Ann'],
            'Math' :
['Marcus','Alex','Glenn','Samuel','Clara','Chelsea'],
            'PE'  : ['Adam','Tyler', 'Alex','Clara'],

            'Social Sciences':
['Abraham','Marcus','Alex','Glenn','Clara'],
```

100% z Lekce 3

Example of running the code:

Osnova

```
~/PythonBeginner/Lesson2 $ python class_stats.py
Students inscribed into all the subjects: {'Alex'}
```

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

REVIEW EXERCISES

- what we want to extract is the **intersection** among all the student listings
- we need to acquire each list from the classes dictionary. We shall do that using the corresponding key inside square brackets (e.g. classes['Biology'])
- in order we can apply **intersection** operation we have to convert each list into a set using `set()` function (e.g. `set(classes['Biology']`)
- once we have sets representing each class, we can perform intersection operation among all the classes and store the result in a variable (in our case variable `common`)
- lastly we just print the result

Note how we enclose the whole intersection operation inside parentheses. This trick allows us to write the expression on multiple lines:

```
common = (set(classes['Biology'])
          & set(classes['Math'])
          & set(classes['PE'])
          & set(classes['Social Sciences'])
          & set(classes['Chemistry']))

print('Students inscribed into all the subjects:', common')
```

100% z Lekce 3

Osnova

REVIEW EXERCISES

Creating dictionary

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY / CREATING DICTIONARY

There are quite a few ways how to create a dictionary - we can use:

1. empty dictionary constructor:

```
>>> d1 = dict()
>>> d1
{}
```

2. empty curly braces:

```
>>> d2 = {}
>>> d2
```

100% z Lekce 3

3. filled with key: value pairs:

Osnova

```
>>> d3 = {'First_Name': 'John','Last_Name': 'Smith','Age':56}
>>> d3
{'First_Name': 'John','Last_Name': 'Smith','Age':56}
```

4. constructor filled with keyword arguments:

```
>>> d4 = dict(First_Name= 'John',Last_Name= 'Smith',Age=56)
>>> d4
{'Age': 56, 'First_Name': 'John', 'Last_Name': 'Smith'}
```

Keys

REVIEW EXERCISES

Dictionary keys **can be only objects of immutable** data type - number, string, tuple, bytes.

```
>>> my_dict = {'name' : 'John', 'age': 32}
>>> my_dict['name']
'John'
```

It's clear then that we **cannot use any mutable** data type as a key:

```
>>> my_dict = {['a','b','c'] : 'name'}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

List `['a','b','c']` is mutable - it means we can add and remove items from it. Dictionary keys, however, have to be constant. We cannot permit a key, which serves as a unique identifier, to be able to be changed. It is like allowing to change your internet banking credentials. The IB system would not be able to identify you anymore.

Fun Fact: While reading python documentation, you may encounter a word **hashable** or

100% z Lekce 3

Why integers as keys?

Osnova

Using integers as keys is unnecessary as in that case we would better **use one of the sequence data types instead**. Items of sequences are by default indexed by (and moreover ordered by) integers.

Integers in dictionary:

```
>>> my_dict = {0 : 'name',1:'age'}
>>> my_dict[1]
'age'
```

Integers in list:

```
>>> data = ['name', 'age']
>>> data[1]
'age'
```

Values

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY / VALUES

Each key can has just one associated value. However, unlike keys, value **can be a mutable** as well as immutable collection of multiple objects if needed (list, tuple, dictionary, set, but also numbers, strings, bytes, bytearray).

```
>>> my_dict = {'volume' : [3,4,7],'surface': [3,4], 'lengths': [3,4,7]}
>>> my_volume = my_dict['volume'][0] * my_dict['volume'][1] *
my_dict['volume'][2]
>>> my_volume
84
```

Above we repeat the list object [3,4,7] twice, and that is absolutely permitted for dictionary values. Any given value can be stored under any number of keys - meaning, **values can repeat in a dictionary, but keys not**. Keys have to be unique identifiers of every key:value pair.

100% z Lekce 3

Another [illegible] where values can repeat themselves is a attendance record stating which de[illegible] last week's lesson:

Osnova

```
>>> lesson1_presence = {'John':True, 'Bob': True, 'Kate' : False, 'Fred'
: True}
```

Insert new value

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY / INSERT NEW VALUE

To insert a new value into a dictionary we assign the value to a variable referring our dictionary. We have to use the square brackets specifying the key we want to associate the value with:

REVIEW EXERCISES

```
>>> my_dict = {}
>>> my_dict['name'] = 'John'
>>> my_dict['surname'] ='Smith'
>>> my_dict
{'name': 'John', 'surname': 'Smith'}
```

Order?

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY / ORDER?

It is important to note that in dictionaries **key-value pairs are not ordered** at least up to Python 3.5. From Python 3.6 dictionary values are already ordered.

Python 3.5

If for example we created a new dictionary as follows in Python 3.5, the key order will be changing:

100% z Lekce 3

```
>>>
Osnova
name': 'Clark', 'age': 34, 'surname': 'Kent'}
```

That way we cannot be sure what key:value pair will be removed when using `dict.popitem()` method:

```
>>> employee.popitem()
('id', 'X5342')
>>> employee.popitem()
('name', 'Clark')
>>> employee.popitem()
('age', 34)
>>> employee.popitem()
('surname', 'Kent')
```

Now we turn off our Python 3.5 terminal and run it again. Then we create the same dictionary as before. However, the order of items is popped out in different order:

REVIEW EXERCISES

```
>>> employee = {'id': 'X5342', 'name': 'Clark', 'surname' : 'Kent', 'age':
34}
>>> employee.popitem()
('surname', 'Kent')
>>> employee.popitem()
('id', 'X5342')
>>> employee.popitem()
('name', 'Clark')
>>> employee.popitem()
('age', 34)
```

Python 3.6+

On the other hand, using Python 3.6 and higher, the order of items will be the same each time we recreate the dict and destroy it.

```
>>> employee = {'id': 'X5342', 'name': 'Clark', 'surname' : 'Kent',
'age': 34}
```

100% z Lekce 3

```
>>> employee.popitem()
('surname', 'Kent')
>>> employee.popitem()
('name', 'Clark')
>>> employee.popitem()
('id', 'X5342')
```

Although the dictionary order is kept in Python 3.6+, it should **not be relied upon**. This feature can change in the future. Also, dictionary is not meant to be used this way. If we need to have ordered collection, we should for instance use a list.

REVIEW EXERCISES

Conversion

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY / CONVERSION

As dictionary is a collection of key:value pairs, the `dict()` constructor arguments have to provide a **pairwise structure** as well. This prerequisite is met, for example, by passing two-dimensional tuple. Dict constructor expects 1 argument to be passed in. Therefore a single collection of data has to be passed in.

Now we will answer to the question: What can and cannot be entered into `dict()` constructor?

CAN

1. In the example below, we have entered 1 tuple containing 3 other two-dimensional tuples

```
>>> dict(((1,2),(3,4),(5,6)))
{1: 2, 3: 4, 5:6}
```

2. Here we are passing a tuple containing only 1 nested tuple (note the comma after the tuple `(1,2)`):

```
>>> dict(((1,2),))
```

100% z Lekce 3

```
>>
```

Osnova

CANNOT

1. Each tuple serves as a key - value pair. If there would be more than 2 items in each tuple, we would get an error:

```
>>> dict(((‘1’,’2’,’3’),(1,2,3)))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: dictionary update sequence element #0 has length 3; 2 is required
```

2. Also one tuple with even number of items would not work:

```
>>> dict((1,2,3,4))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot convert dictionary update sequence element #0 to a sequence
```

3. The `dict()` constructor expects **one object as input** and therefore passing multiple tuples inside won't work:

```
>>> dict(('1','2'),(1,2))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: dict expected at most 1 arguments, got 2
```

4. The first item in the input tuple is made a key and the second is stored in the value part of the pair. Therefore the first item has to be of immutable data type. For example, list `[1]` is **not immutable** - therefore an error is raised:

```
>>> dict([[[1],2],[3,4]])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

100% z Lekce 3

Osnova

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY / NESTING

We can nest dictionaries inside dictionaries as values (not keys). This allows for modelling of more complex categorizations. In the example below, we can see dictionary `{'Job Title': 'System Admin', 'Level' : 3}` nested under the key `'Job'`. Or also the `'Address'` key refers to a another nested dictionary:

```
my_db = {'Name': 'John Smith',
         'Age': 34,
         'Address': {'Street': 'Main',
                     'Street #': 241,
                     'City': 'Boston',
                     'Country': 'Venezuela'},

         'Job': {'Job Title': 'System Admin',
                 'Level' : 3}
        }
```

If we had a database of employees stored in this way in a dictionary and we wanted to know, what is this employee's home country, we would be very happy to have the address key to contain another dictionary with more specific categorization:

```
>>> my_db['Address']['Country']
'Venezuela'
```

Code Task

1. Try to access the value stored under the `'Level'` key, which is stored under the `'Job'` key inside the `my_db` dictionary.
2. Try to concatenate the whole address into one string. City and Country should be separated by comma.

The result could look like this:

100% z Lekce 3

```
# Database
Osnova      me': 'John Smith',
            ge': 34,
           'Address': {'Street': 'Main',
               'Street #': 241,
               'City': 'Boston',
               'Country': 'Venezuela'},

           'Job': {'Job Title': 'System Admin',
                   'Level' : 3}
}

# 1

# 2

# Print
```

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

```
# Database
my_db = {'Name': 'John Smith',
```

100% z Lekce 3

Osnova

```
                    'City': 'Boston',
                    'Country': 'Venezuela'},

         'Job': {'Job Title': 'System Admin',
                 'Level' : 3}
        }


# 1
level = my_db['Job']['Level']

# 2
address = my_db['Address']
formatted_address = (str(address['Street #']) + ' ' + address['Street'] + ' ' + 'Street, ' +
                    address['City'] +
                    ', ' + address['Country'])

# Print
print(level)
print(formatted_address)
```

Accessing values

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY / ACCESSING VALUES

Dictionary methods, as we will see, many times duplicate the functionality, we have already seen performed with the square brackets operation as accessing, updating or removing values. However, the main benefit in using them is that they handle situations when keys do not exists and they do not raise errors.

In the following examples we will use the variable `my_db` that contains the following information:

100% z Lekce 3

```
 'A
      Osnova
  A          eet': 'Main',
             'Street #': 241,
             'City': 'Boston',
             'Country': 'Venezuela'},
 'Job': {'Job Title': 'System Admin',
         'Level' : 3}
}
```

Method .get()

One of the methods that handle such situation is the method `.get()` .

Syntax: `my_db.get(key[,default_value=None])`

Arguments accepted by the method are key and optional default value, for case if key is not present in the dictionary. That way the error is not raised if the key does not exist in the dictionary.

```
>>> my_db.get('Address')
{'City': 'Boston',
 'Country': 'Venezuela',
 'Street': 'Main',
 'Street #': 241
}
```

Why do we need a special method for something that already exists as `my_db['Address']` ? The reason is, that the get() method allows us to avoid errors in our programs - if the key is not found, the default value specified as a second input argument is returned:

```
>>> my_db.get('Salary',0)
0
```

Code Task

- Try to use the `get()` method to retrieve value stored under they key `'Birth'` . If such key

- T[illegible]ve [illegible]e stored under the key `'Age'`, if no such key found, value 0 should be

Osnova

```
1
```

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

100% z Lekce 3

Osnova

```
            'Address': {'Street': 'Main',
                        'Street #': 241,
                        'City': 'Boston',
                        'Country': 'Venezuela'},

            'Job': { 'Job Title': 'System Admin',
                     'Level' : 3}
          }

birth =  my_db.get('Birth', '0.0.0000')
age = my_db.get('Age', 0)

print(birth, age)
```

REVIEW EXERCISES

Updating a dictionary

We can enter new dictionaries or just key-value pairs inside a dictionary by using the `update()` method. The expected input of the `update()` method has to be another dictionary or so called keyword arguments.

Syntax: `my_db.update([other])`

We will continue to work with a variable `my_db` that contains the following data:

```
>>> my_db
{'Name': 'John Smith',
 'Age': 34,
 'Address': {'Street': 'Main',
             'Street #': 241,
             'City': 'Boston',
             'Country': 'Venezuela'},
```

100% z Lekce 3

Osnova

Update method

First, let's updated the dictionary inside `my_db` variable with key 'Performance' by passing a dictionary into `update()`

```
>>> my_db.update({'Performance': {'Q1': 1, 'Q2':1, 'Q3':2, 'Q4':1}})
>>> my_db
{'Name': 'John Smith',
 'Age': 34,
 'Address': {'Street': 'Main',
             'Street #': 241,
             'City': 'Boston',
             'Country': 'Venezuela'},
 'Job': {'Job Title': 'System Admin','Level' : 3},
 'Performance' : {'Q1': 1, 'Q2':1, 'Q3':2, 'Q4':1}
}
```

Update keyword arguments

Second, we update existing key 'Name' by so called keyword arguments (`key = value`):

```
>>> my_db.update(Name ='John E. Smith')
>>> my_db
{'Name': 'John E. Smith',
 'Age': 34,
 'Address': {'Street': 'Main',
             'Street #': 241,
             'City': 'Boston',
             'Country': 'Venezuela'},
 'Job': {'Job Title': 'System Admin',
         'Level' : 3},
 'Performance' : {'Q1': 1, 'Q2':1, 'Q3':2, 'Q4':1}
}
```

100% z Lekce 3

Osnova

Try to add the manager name to `my_db` dictionary under the key `'Manager'` and value `'Samuel, Hunt'` using the `update()` method.

```
1
```

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

100% z Lekce 3

Osnova

```
my_db = {'Name': 'John E. Smith',
         'Age': 34,
         'Address': {'Street': 'Main',
                     'Street #': 241,
                     'City': 'Boston',
                     'Country': 'Venezuela'},

         'Job': {'Job Title': 'System Admin',
                 'Level' : 3},
         'Performance' : {'Q1': 1, 'Q2':1, 'Q3':2, 'Q4':1}
        }

my_db.update(Manager = 'Samual, Hunt')

print(my_db)
```

REVIEW EXERCISES

Removing items

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY / REMOVING ITEMS

To remove key-value pairs (items) from a dictionary, we can use the `del` keyword as follows: `del my_dict[key]`

In the following section we will keep working with dictionary `my_db` :

```
>>> my_db
{'Name': 'John Smith',
 'Age': 34,
 'Address': {'Street': 'Main',
             'Street #': 241,
             'City': 'Boston',
             'Country': 'Venezuela'},
 'Job': {'Job Title': 'System Admin','Level' : 3},
```

100% z Lekce 3

Therefore, if we wanted to get rid of the `Performance` key in our dictionary, we can write down the following:


```
>>> del my_db['Performance']
```

Now if we print the content of `my_db` variable, we can see `Performance` key no more:

```
>>> my_db
{'Name': 'John Smith',
 'Age': 34,
 'Address': {'Street': 'Main',
             'Street #': 241,
             'City': 'Boston',
             'Country': 'Venezuela'},
 'Job': {'Job Title': 'System Admin','Level' : 3}
}
```

Removing Data

We have some additional methods for removing data from dictionary.

Method .pop()

The `.pop()` method removes the specified key and returns the associated value. If the key does not exist, we can specify the default value that should be returned in such case.

The syntax is `my_db.pop(key[,default_value])` and an example would be:

```
>>> my_db.pop('FTE',1)
1
```

The key 'FTE' does not exist in our dictionary, so the value 1 has been returned - we can later on

Let's add the 'FTE' category into our dictionary and then immediately pop it - if we try to retrieve it, we will get error.


```
>>> my_db['FTE'] = 0.5
>>> my_db.pop('FTE',1)
0.5
>>> my_db['FTE']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'FTE'
```

Method .popitem()

This method removes some (arbitrary) key-value pair from the dictionary and returns the pair as a tuple. The specific pair cannot be specified, therefore this method is used for continual destruction of a dictionary.

The syntax is `my_db.popitem()` and an example:

```
>>> my_db.popitem()
('Address', {'Country': 'Venezuela', 'City': 'Boston', 'Street': 'Main',
'Street #': 241})
```

If there are no more items (key-value pairs) inside the dict, KeyError is raised.

```
>>> my_db = {}
>>> my_db.popitem()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'popitem(): dictionary is empty'
```

The `.popitem()` method can be used, when we gradually want to empty the whole dictionary and every time we want to take one key:value pair for processing. This will be more usefully demonstrated with so called loops.

Method .clear()

The s[illegible] d[illegible]`lear()` and an example:

Osnova

```
>>> my_db.clear()
>>> my_db
{}
```

Copying dictionary

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY / COPYING DICTIONARY

To create a copy (a new object) of a dictionary, we have to use its `.copy()` method. It returns so called shallow copy of the original dictionary. Shallow copy means, that the nested mutable collections (any dictionaries or lists or sets) are still the same objects and so if we will change them in any way, the original instance of the dictionary will also reflect these changes.

REVIEW EXERCISES

In our example below we are working with the dictionary `my_db` :

```
my_db = { 'Name': 'John E. Smith',
          'Age': 34,
          'Address': {'Street': 'Main',
                      'Street #': 241,
                      'City': 'Boston',
                      'Country': 'Venezuela'},

          'Job': {'Job Title': 'System Admin','Level' : 3},
          'Performance' : {'Q1': 1, 'Q2':1, 'Q3':2, 'Q4':1}
        }
```

Now we copy `my_db` to `new_db` :

```
>>> new_db = my_db.copy()
>>> new_db
{'Name': 'John E. Smith',
 'Age': 34,
```

100% z Lekce 3

Osnova

```
                'City': 'Boston',
                'Country': 'Venezuela'},
 'Job': {'Job Title': 'System Admin','Level' : 3},
 'Performance' : {'Q1': 1, 'Q2':1, 'Q3':2, 'Q4':1}
}
```

Watch out for mutable values

Keys `'Performance'`, `'Job'` and `'Address'` refer to mutable data types. Therefore if we change them in `new_db`, **the change will be reflected in** `my_db` as well, because `new_db` is just a shallow copy of `my_db`:

```
>>> del new_db['Performance']['Q1']
>>> new_db
{'Name': 'John E. Smith',
 'Age': 34,
 'Address': {'Street': 'Main',
             'Street #': 241,
             'City': 'Boston',
             'Country': 'Venezuela'},
 'Job': {'Job Title': 'System Admin',
         'Level' : 3},
 'Performance' : {'Q2':1, 'Q3':2, 'Q4':1}
}
>>> my_db
{'Name': 'John E. Smith',
 'Age': 34,
 'Address': {'Street': 'Main',
             'Street #': 241,
             'City': 'Boston',
             'Country': 'Venezuela'},
 'Job': {'Job Title': 'System Admin',
         'Level' : 3},
 'Performance' : {'Q2':1, 'Q3':2, 'Q4':1}
}
```

100% z Lekce 3

We could create a copy of the nested dictionaries:

Osnova

```
>>> address = my_db['Address'].copy()
>>> address
{'Street': 'Main',
 'Street #': 241,
 'City': 'Boston',
 'Country': 'Venezuela'
}
```

However, if we were to delete an item from that dictionary, the original dictionary, when the `'Address'` was nested would remain **the same**.

```
>>> del address['Street']
>>> address
{'Street #': 241,
 'City': 'Boston',
 'Country': 'Venezuela'
}
>>> my_db
{'Name': 'John E. Smith',
 'Age': 34,
 'Address': {'Street': 'Main',
             'Street #': 241,
             'City': 'Boston',
             'Country': 'Venezuela'},
 'Job': {'Job Title': 'System Admin','Level' : 3},
 'Performance' : {'Q1': 1, 'Q2':1, 'Q3':2, 'Q4':1}
}
```

What we've been working with here is so called **shallow copy**. There's also a **deep copy**. For more information you can check out [Python documentation] on this topic.

Dictionary Views

100% z Lekce 3

Dictionary views is a new concept in Python 3 compared to Python 2. Dictionary views are three objects returned by the following three dictionary methods:

Osnova

- `.keys()` - returns objects of **type dict_keys** that refer to all top level keys in dictionary
- `.values()` - returns objects of **type dict_values** that refer to all top level values in dictionary
- `.items()` - returns objects of **type dict_items** that refer to all tuples of top level key-value pairs in dictionary

In order to see the contents of keys, values and items objects, we need to **convert them into list or a tuple**

Bellow, you'll have a task using the following dictionary:

```
my_db = {   'Name': 'John Smith',
            'Age': 34,
            'Address': {'Street': 'Main',
                'Street #': 241,
                'City': 'Boston',
                'Country': 'Venezuela'},

            'Job': {'Job Title': 'System Admin',
                    'Level' : 3}
        }
```

Why to use views?

We will use the dictionary views when working with **for loop**, but will get to the in the future lesson.

Code Task

1. get the list of the keys from `my_db`
2. get the list of the values from `my_db`
3. get the list of the items as well :)

100% z Lekce 3

Osnova

Try to get **sorted** version of the values.

```
my_db = {   'Name': 'John Smith',
            'Age': 34,
            'Address': {'Street': 'Main',
                'Street #': 241,
                'City': 'Boston',
                'Country': 'Venezuela'},

            'Job': {'Job Title': 'System Admin',
                    'Level' : 3}
        }

# 1.

# 2.

# 3.
```

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

100% z Lekce 3

Osnova

```
            'Age': 34,
            'Address': {'Street': 'Main',
                'Street #': 241,
                'City': 'Boston',
                'Country': 'Venezuela'},

            'Job': {'Job Title': 'System Admin',
                    'Level' : 3}
}

# 1.
print(list(my_db.keys()))
# 2.
print(list(my_db.values()))
# 3.
print(list(my_db.items()))
```

REVIEW EXERCISES

Note that in the case of **keys** you should know that `list(my_db)` will have the same output as `list(my_db.keys())` :

```
['Name', 'Age', 'Address', 'Job']
```

Bonus

```
# 1.
print(sorted(my_db.keys()))
# 2.
print(sorted(my_db.values()))
# 3.
print(sorted(my_db.items()))
```

Note that in the case of `sorted(my_db.values())` we get an error, because we cannot sort object that includes integers as well as strings:

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'int' and 'str'
```

100% z Lekce 3

Osnova

REVIEW EXERCISES

Creating Dictionary

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY + / CREATING DICTIONARY

Alternatively, we could use the method `.fromkeys()` . It returns a new dictionary with keys taken from the sequence argument and values set to value specified as optional argument. All the keys will have the same value assigned.

The method can be used **only directly on dict class** and **not on a specific dictionary object**. Therefore this method will be always written as `dict.fromkeys()` .

The syntax is `dict.fromkeys(sequence[,value])` and an example:

```
>>> d = dict.fromkeys(('Account1','Account2','Account3'),0)
```

100% z Lekce 3

One look of functional tools

Osnova

Let's consider this paragraph to be a bonus material on how to create a dictionary - in the future, we will use it more.

Using `zip()` function on 2 iterables, all enclosed in dict constructor:

```
>>> dict(zip(('Name','Age'),('John',45)))
{'Age': 45, 'Name': 'John'}
```

```
>>> dict(zip(range(10),range(10)))
{0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9}
```

Zip function connects together into key:value pairs objects at the same index in both sequeces - that way we get a tuple of 2-item tuples from which `dict()` creates a dictionary.

Don't worry about this function too much though. This is only a demonstration of other interesting ways of working with dictionary :)

REVIEW EXERCISES

Dictionary methods

PYTHON ACADEMY / 3. DICTIONARIES & SETS / DICTIONARY + / DICTIONARY METHODS

Finally, let's check the summary of frequently used methods, which might help us while working with dictionaries.

Operation	Syntax	Description	Example	Output
Update the dictionary	`.update()`	Adds elemnts from other dictionary or iterable pairs (tuples).	`store = {'butter': 2.35, 'bread': 0.95}; new = {'milk': 1.15};`	`{'milk': 1.15, 'bread': 0.95, 'butter': 2.35}`

100% z Lekce 3

Osnova

Syntax	Description	Example	Output
Insert missing keys with values	`.setdefault()`	Insert key with a value, if key is not present	`person = {'name':'Martin'}; person.setdefault('age', 25)`	`{'age': 25, 'name': 'Martin'}`
Get value	`.get()`	Returns value of the key.	`dict = {'a': 1, 'b': 2}; dict.get('b')`	`2`
Remove and return value	`.pop()`	Removes and returns the value of the chosen key.	`dict = {'a': 1, 'b': 2}; dict.pop('a')`	`1`
Remove and return pair	`.popitem()`	Removes and returns a pair from the dictionary.	`dict = {'a': 1, 'b': 2}; dict.popitem()`	`('a', 1)`
Remove items	`.clear()`	Removes all items.	`dict = {'a': 1, 'b': 2}; dict.clear()`	`{}`
Copy dictionary	`.copy()`	Returns shallow copy of a dictionary.	`dict = {'a': 1, 'b': 2}; dict.copy()`	`{'a': 1, 'b': 2}`
Create from keys	`.fromkeys()`	Returns a new dictionary with keys from given sequence.	`dict = {}; list = ["A","B","C"]; dict.fromkeys(list)`	`{'C': None, 'B': None, 'A': None}`
View keys	`.keys()`	Returns view object of all keys.	`dict = {'a': 1, 'b': 2}; dict.keys()`	`['a', 'b']`
View values	`.values()`	Returns view object of all	`dict = {'a': 1, 'b': 2};`	`[1, 2]`

REVIEW EXERCISES

100% z Lekce 3

Osnova	Syntax	Description	Example	Output
View pairs	`.items()`	Returns view of (key, value) pair.	`dict = {'a': 1, 'b': 2}`; `dict.items()`	`[('a', 1), ('b', 2)]`

REVIEW EXERCISES

Introduction

PYTHON ACADEMY / 3. DICTIONARIES & SETS / SETS / INTRODUCTION

100% z Lekce 3

[illegible] it has been added.

Osnova

There are **2 types** of set data in Python - **set & frozenset**. The difference among the two is that set is mutable collection whereas frozenset is immutable. Otherwise the essence of both is the same.

Use

As sets are not ordered and do not contain any keys, therefore we **cannot use**:

- indexing,
- slicing,
- but neither repetition,
- or concatenation.

REVIEW EXERCISES

Therefore the **main use** of sets is:

- membership testing,
- assuring the uniqueness of each item in a collection.

Syntax

Items in set are enclosed in **curly braces** - similarly to a dictionary. However, there are **no key:value pairs**, what distinguishes dictionaries from sets. Dictionary keys have to be also **unique** as the set items and maybe therefore the similarity with curly braces.

Type	Syntax
Set	`{'a','b','c'}`
Frozenset	`frozenset({'a', 'b', 'c'})`

Creating set

100% z Lekce 3

Empty sets can be created by using the **set constructor** - `set()`

Osnova

```
>>> my_set = set()
>>> type(my_set)
<class 'set'>
```

```
>>> my_set = frozenset()
>>> type(my_set)
<class 'frozenset'>
```

However, we **cannot use curly braces** `{}` to create an empty set. We would be creating a new dictionary:

```
>>> data = {}
>>> type(data)
<class 'dict'>
```

REVIEW EXERCISES

Non-empty sets (not frozensets) can be created by placing a comma-separated elements inside the curly braces, for example:

```
>>> my_set = {'John', 'Rob'}
>>> type(my_set)
<class 'set'>
```

Example of use

As sets are collections of unique objects, we can use them to determine for example what unique letters are present in a string:

```
>>> set('Popocatepetl')
{'P', 'a', 'c', 'e', 'l', 'o', 'p', 't'}
```

Or maybe we have a sequence of integers mixed with strings, that also works for sets:

```
>>> s = set([1,2,4,2,5,6,4,3,3,'a'])
>>> s
{1, 2, 3, 4, 5, 6, 'a'}
```

100% z Lekce 3

Osnova

Operations

If we want to keep and change a collection of unique elements, we will surely need to know, how to add and remove items from it. We know we need a set to keep a collection of unique elements.

By using sets, we do not have to care, whether currently added element is already present in the collection, because set will always keep only one instance of that item.

Adding items to a set

To add a new element to a set we have to use `add()` method: `set1.add(item)`

REVIEW EXERCISES

Example where the item being added is not present yet in the set:

```
>>> names = {'Marcus', 'Alex'}
>>> names.add('Oliver')
>>> names
{'Marcus', 'Oliver', 'Alex'}
```

Example where the item being added is already present in the set:

```
>>> names = {'Marcus', 'Oliver', 'Alex'}
>>> names.add('Oliver')
>>> names
{'Marcus', 'Oliver', 'Alex'}
```

Set takes care we do not have duplicates in it.

Removing items from a set

To remove an element from a set we have to use `discard()` method: `set1.discard(item)`

```
>>> names = {'Marcus', 'Oliver', 'Alex'}
>>> names.discard('Oliver')
```

100% z Lekce 3

Osnova

operations

PYTHON ACADEMY / 3. DICTIONARIES & SETS / SETS / COMMON SET OPERATIONS

Since sets store only the **unique elements** they are really useful when we have f.e. a list and we'd like to know if:

1. There is a certain element in that collection:

```
>>> my_list = [1,1,2,3,4,5,5]
>>> 5 in set(my_list)
True
```

Notice that we convert the list into a set as we don't need all the duplicate elements of `my_list` . We only need the unique elements to establish whether or not the element is part of our list. We should already know that this operation is called **membership testing** - we use the keyword `in` .

REVIEW EXERCISES

2. How many unique elements a collection has:

```
>>> len(set(my_list))
5
```

Again we convert the list into a set and then we check its length with the function `len()` .

Set Operations

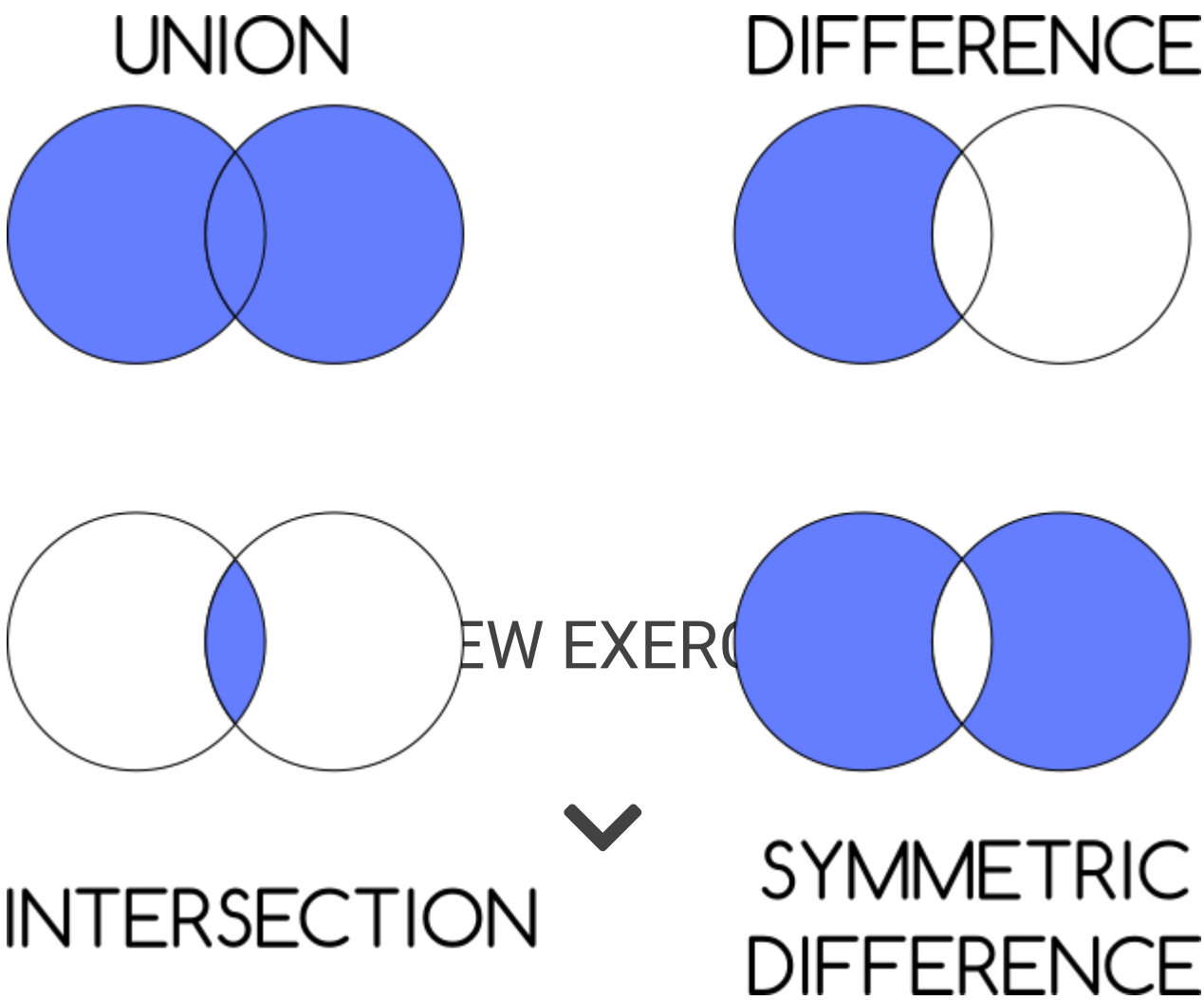
PYTHON ACADEMY / 3. DICTIONARIES & SETS / SETS / SET OPERATIONS

Sets can be combined in a number of different ways to produce another set. We normally want to find out, what items:

- are in all sets, we combine in the operation (**union**)
- what items are in one but not in the other set (**difference**)

100% z Lekce 3

Osnova



Set Operations - Union

PYTHON ACADEMY / 3. DICTIONARIES & SETS / SETS / SET OPERATIONS - UNION

The union operation **"unites"** the elements of all the sets included in the union operation. It returns a new set. We use the `|` operator to perform union on sets: `set1 | set2 | set3 | ...`

```
>>> set('Hello') | set('Yellow') | set ('Fellow')
{'l', 'e', 'F', 'H', 'w', 'o', 'Y'}
```

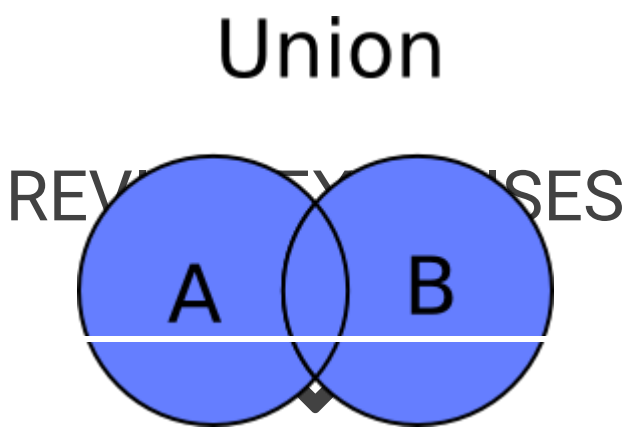
100% z Lekce 3

```
>>> [illegible] | 'Yellow' | 'Fellow'
Osnova
Traceback [illegible] recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for |: 'str' and 'str'
```

To execute union operation, we can also use a **method**: `set1.union(others)`

Example:

```
>>> set('Hello').union(set('Yellow'),set ('Fellow'))
{'l', 'e', 'F', 'H', 'w', 'o', 'Y'}
```



Set Operations - Difference

PYTHON ACADEMY / 3. DICTIONARIES & SETS / SETS / SET OPERATIONS - DIFFERENCE

The **difference** operation produces a set of items, that are present in one set, but not in the other. It returns a new set. We use the `-` operator to perform difference on sets: `set1 - set2 - set3 ...`

```
>>> set('Hello') - set('Yellow') - set('Fellow')
{'H'}
```

We cannot use the **difference** operator with other data type than sets:

```
[illegible]
```

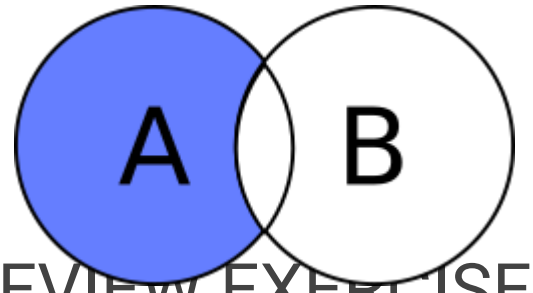
100% z Lekce 3

```
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'str' and 'str'
```


To execute difference operation, we can also use a **method**: `set1.difference(others)`

Example:

```
>>> set('Hello').difference(set('Yellow'),set ('Fellow'))
{'H'}
```



Set Operations - Symmetric Difference

The **symmetric difference** operation produces a new set with elements in either set but not in both. We use the `^` operator to perform symmetric difference on sets: `set1 ^ set2`

```
>>> set('Hello') ^ set('Yellow')
{'H', 'w', 'Y'}
```

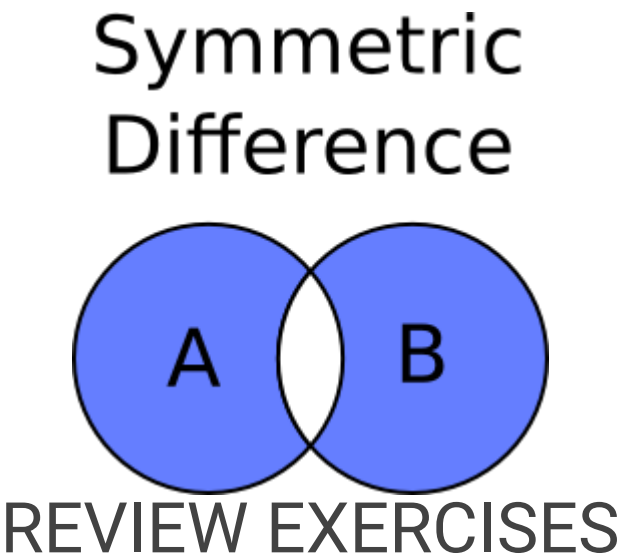
We cannot use the **symmetric difference** operator with other data type than sets:

```
>>> 'Hello' ^ 'Yellow'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for ^: 'str' and 'str'
```

To execute symmetric difference operation, we can also use a **method**:

Osnova

```
>>> set('Hello').symmetric_difference(set('Yellow'))
{'H', 'w', 'Y'}
```



Set Operations - Intersection

PYTHON ACADEMY / 3. DICTIONARIES & SETS / SETS / SET OPERATIONS - INTERSECTION

The intersection operation produces a new set containing only **elements that are shared** by all sets implicated in the operation. We use the `&` operator to perform intersection on sets: `set1 & set2 & set3 & ...`

```
>>> set('Hello') & set('Yellow') & set('Fellow')
{'e', 'o', 'l'}
```

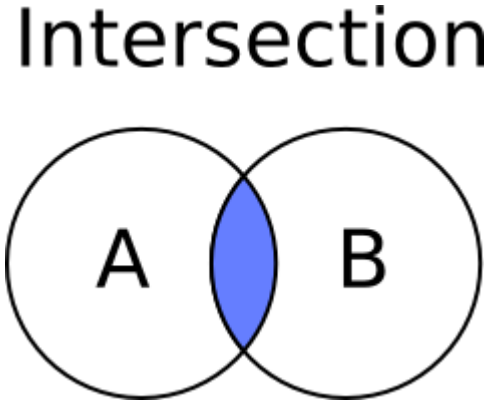
We **cannot use** the intersection operator with other data type than sets:

```
>>> 'Hello' & 'Yellow' & 'Fellow'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for &: 'str' and 'str
```

To execute intersection operation, we can also use a **method**: `set1.intersection(others)`

100% z Lekce 3

```
>>> set('Hello') intersection(set('Yellow'),set ('Fellow'))
{'e', 'l', 'o'}
```

Set Operations - Subset

An extra set operation is subset. This is when we test whether every element in **set1 is in the set2**. For this we can use less than `<` and equal to `=` operators:

```
>>> set('Hello') <= set('Yellow H')
True
```

Subset operation tests the relation in **only one way** - whether all elements of set1 are included in set2 - but this does not imply that all elements of set2 are in set 1

```
>>> set('Hello') >= set('Yellow H')
False
```

If we wanted to make clear in our code, that we are comparing two sets, we can also use a set method `issubset()` : `set1.issubset(set2)`

```
>>> set('Hello').issubset(set('Yellow H'))
True
```

If we wanted to get a **proper subset** and make sure, that all elements of set1 are present in set2,

Osnova ns - Disjoint

Lastly, to find out, whether two sets have **no elements in common**, we can use the intersection operator. If the result of intersection of two sets, is an empty set, then we call this relation to be disjoint: `len(set1 & set2) == 0`

Example:

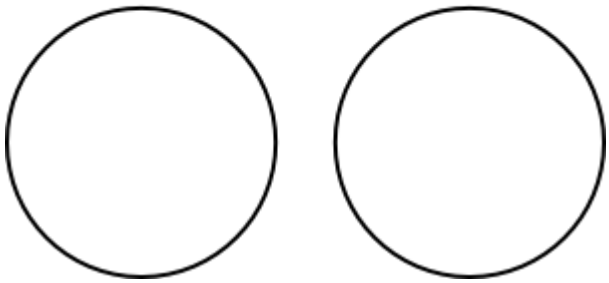
```
>>> set((1,2,3,4)) & set((6,7,8))
set()
>>> len({1,2,3,4} & {6,7,8})
0
```

There is also a method we can use to find out, whether 2 sets have any elements in common: `set1.isdisjoint(set2)`

REVIEW EXERCISES

```
>>> {1,2,3,4}.isdisjoint({6,7,8})
True
```

Disjoint



100% z Lekce 3

Osnova

REVIEW EXERCISES

Dictionary

PYTHON ACADEMY / 3. DICTIONARIES & SETS / QUIZ / DICTIONARY

1/17

What is the difference among dictionary and set?

A. Dictionary does not have any unique values

B. Sets are not composed of pairwise values

C. Sets are ordered

D. Sets are immutable

100% z Lekce 3

Osnova

PYTHON ACADEMY / 3. DICTIONARIES & SETS / QUIZ / SET

1/5

Select the correct set definitions:

A. Sets are ordered

B. Items in sets can repeat

REVIEW EXERCISES

C. Sets contain only unique items

100% z Lekce 3

Osnova

Delete value

PYTHON ACADEMY / 3. DICTIONARIES & SETS / HOME EXERCISES - DICTIONARY / DELETE VALUE

At the beginning we have a dictionary: `myNewDict = {'m': 12345, 'n': 32145, 'o': 54321, 'p': 23232, 'q': 43210, 'r': 13579}`

Your task is to complete the following instructions:

- First of all we want to get the key which has alphabetically the maximal value - `maximalValueofKey`,
- second step is print that value out,
- if any value in our dictionary is greater than the value of our maximal key (`maximalValueOfKey`), we want to delete the whole item under the key `maximalValueOfKey`
- finally we want to print our new modified dictionary.

REVIEW EXERCISES

Online Python Editor

```
1
```

100% z Lekce 3

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see how we wrote the code.

REVIEW EXERCISES

Click to see our solution

```
# Let's define a new dictionary
myNewDict = {
    'm': 12345,
    'n': 32145,
    'o': 54321,
    'p': 23232,
    'q': 43210,
    'r': 13579
    }

# we want to get the highest value in keys
# and also print it
maximalValueOfKey = max(myNewDict.keys())
print('The maximal value has the key: ' + maximalValueOfKey)

# the next step is to delete the key if its value is greater than
```

100% z Lekce 3

```
    del myNewDict[maximalValueOfKey]
```

Osnova

```
# finally what is the current form of our dictionary?
print(myNewDict)
```

Password check 2

PYTHON ACADEMY / 3. DICTIONARIES & SETS / HOME EXERCISES - DICTIONARY / PASSWORD CHECK 2

We have this dictionary:

```
data = {'user1': 'password1', 'Mark': '1234', 'Danny': 'qwert'}
```

REVIEW EXERCISES

In this task, we will try to verify if the user enters a password that belongs to his account.

The output should looks like:

```
~/PythonBeginner/Lesson3 $ python verify.py
Please enter username: Mark
Please enter password: 1234
Permission continue, GRANTED!
```

If the password will be type incorrectly or is incorrect in general:

```
~/PythonBeginner/Lesson3 $ python verify.py
Please enter username: Mark
Please enter password: 444
Password or username are WRONG!
```

Online Python Editor

```
1
```

100% z Lekce 3

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

- we need to begin with two inputs from user
- next step is to check if the typed password belongs to the correct username from the dictionary
- if it is true, we want to print answer about it
- if it is not the correct pair of username and password, we want to inform the user

```
# our dictionary with data
```

100% z Lekce 3

Osnova

```
    'Mark': '1234',
    'Jan': 'qwert',
            }

# we want to ask user for username and password
Username    = input('Please enter the username: ')
Password    = input('Please enter the password: ')

# two conditions for evaluating the inputs
if data.get(Username) != Password:
    print('Password or username is wrong')

elif data.get(Username) == Password:
    print('Permission granted')
```

REVIEW EXERCISES

100% z Lekce 3

Osnova 'nique

Now let's practice what we've learned about sets.

We will work with these two strings:

```
String01 = 'Bratislava'
String02 = 'Budapest'
```

Write a script that will find and print only elements using a suitable operator or method:

1. Common - which have `String01` and `String02` common.
2. Unique - which charaters are present in `String01` but not in `String02` .

REVIEW EXERCISES

The output could look like this:

```
~/PythonBeginner/Lesson3 $ python common_unique_chars.py
Common characters: {'s', 'B', 't', 'a'}
Unique characters: {'i', 'r', 'l', 'v'}
```

Online Python Editor

1

100% z Lekce 3

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution REVIEW EXERCISES

- we will need to create a new variables where we want to save our results
- for each operation a separate variable
- finally we need to print there results

```
# our inputs
String01 = 'Bratislava'
String02 = 'Budapest'

# our operations
Common      = set(String01) & set(String02)
Unique      = set(String01) - set(String02)

# print section
print('Common characters: ' + str(Common))
print('Unique characters: ' + str(Unique))
```

100% z Lekce 3

Osnova **All**

Now, we will work once again with these two strings:

```
String01 - 'Bratislava'
String02 - 'Budapest'
```

Write a script that will find and print only the elements using a suitable operator or method:

1. Difference - which `String01` and `String02` do not share. In other words, the elements that are located in `String01`, `String02`, but not in both.
2. All - which `String01` and `String02` share and do not share - all elements

The output could look like this:

REVIEW EXERCISES

```
~/PythonBeginner/Lesson3 $ python different_all_chars.py
Different characters: {'d', 'v', 'u', 'r', 'i', 'e', 'p', 'l'}
All characters: {'d', 'v', 'e', 'B', 's', 'a', 'u', 't', 'r', 'i', 'p',
'l'}
```

Online Python Editor

1

100% z Lekce 3

Osnova

spustit kód

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

REVIEW EXERCISES

- we will need to create a new variables where we want to save our results
- for each operation a separate variable
- finally we need to print there results

```
# our inputs
String01 = 'Bratislava'
String02 = 'Budapest'

# our operations
Difference  = set(String01) ^ set(String02)
All         = set(String01) | set(String02)

# print() section
print('Different characters: ' + str(Difference))
print('All characters: ' + str(All))
```

100% z Lekce 3

Osnova

REVIEW EXERCISES