School:		
Date:	**Teacher‘s name:**	
Grade:	**Number present:**	**absent:**
Topic of the lesson: Sequences: strings, lists, dictionaries		
Learning objective(s) that this lesson is contributing to	Introduce sequence design: strings, lists, dictionaries Showing the principles of the sequence: lines, lists, dictionaries	
Lesson objectives	**All learners will be able to**: • Know the sequence constructions: strings, lists, dictionaries and use in programming **Most learners will be able to**: • Distinguish between designs, work sequences: strings, lists, dictionaries and use in programming **Some learners will be able to:** • Compose programs using a sequence: lines, lists, dictionaries	
Assessment Criteria	**Owns** the principles of the sequence: lines, lists, dictionaries **Able** to make simple programs using sequences: strings, lists, dictionaries	
Value links	Spiritual development, respect for each other, mutual understanding	
Previous learning	Students work on their level of programming	
Cross curricular links	maths	

Time	**Planned activities**	**Resources**
Beginning **2 min**	**Organizing time** **Greeting students.** **Announcement of the lesson topic, learning objectives, joint definition of lesson objectives and assessment criteria**	slide
Middle **10 min** 5 min	**Go to the topic** **Grouping.** **Discussion with the class.** **“Why did you come together that way?”** **II. Generalization and systematization of knowledge.** Oral frontal survey using presentation. In the Python programming language, dictionaries (type dict) are another kind of data structure along with lists and tuples. A dictionary is a **mutable** (like a list) **unordered** (as opposed to strings, lists, and tuples) set of key-value elements. "Unordered" means that the sequence the location of the pairs is not important. The programming language does not take it into account, as a result of which it is impossible to access elements by indexes. In other languages, structures similar to dictionaries are called differently. For example, in Java, such a data type is called a mapping. To make the idea of the dictionary more understandable, we draw an analogy with a conventional dictionary, for example, English-Russian. For every English word	

16 min

in such a dictionary there is a Russian translation word: cat - cat, dog - dog, table - table, etc. If you describe the English-Russian dictionary using Python, then English words can be made keys, and Russian words can be made values:

{'cat': 'cat', 'dog': 'dog', 'bird': 'bird', 'mouse': 'mouse'}

Pay attention to braces, it is with their help that a dictionary is defined. The syntax of the dictionary in Python is described by the following scheme:

{ключ : значение, ключ : значение, ключ : значение

Often, when a dictionary is displayed, the sequence of key-value pairs does not match the way it was entered:

```
>>> a = {'cat': 'кошка', 'dog': 'собака', 'bird': 'птица', 'mouse': 'мышь'}
>>> a
{'dog': 'собака', 'cat': 'кошка', 'bird': 'птица', 'mouse': 'мышь'}
```

Since the order of the pairs is not important in the dictionary, the interpreter displays them as it suits it. Then how to get access to a certain element if indexing is not possible in principle? In the dictionary, values are accessed by keys, which are enclosed in square brackets (similar to list indexes):

```
>>> a['cat']
'кошка'
>>> a['bird']
'птица'
```

Dictionaries, like lists, are a mutable data type: it is possible to modify, add and delete elements (key: value pairs). Initially, you can create a dictionary empty (for example, d = {}) and then fill it with elements. Adding and changing has the same syntax: dictionary [key] = value. The key can be either already existing (then the value changes), and new (adding a dictionary item). Removing an element is done using the Python built-in del operator.

```
>>> a['elephant'] = 'бегемот' # добавляем
>>> a['table'] = 'стол' # добавляем
>>> a
{'dog': 'собака', 'cat': 'кошка', 'mouse': 'мышь', 'bird': 'птица', 'table': 'стол',
'elephant': 'бегемот'}
>>> a['elephant'] = 'слон' # изменяем
>>> del a['table'] # удаляем
>>> a
{'dog': 'собака', 'cat': 'кошка', 'mouse': 'мышь', 'bird': 'птица', 'elephant': 'слон'}
```

The dictionary cannot have two elements with the same keys. However, different keys may have the same values.

The key can be any immutable data type. Value is any data type. Values of dictionaries may well be structures, for example, other dictionaries or lists.

5 min

```
>>> nums = {1: 'one', 2: 'two', 3: 'three'}
>>> person = {'name': 'Tom', 1: [30, 15, 16], 2: 2.34, ('ab', 100): 'no'}
```

Enumerating dictionary items in a for loop
Vocabulary elements are iterated over in a for loop as well as elements of other complex objects. However, by default, only keys are retrieved:

```
>>> nums
{1: 'one', 2: 'two', 3: 'three'}
>>> for i in nums:
...     print(i)
...
1
2
3
```

But with the keys you can always get the values:

```
>>> for i in nums:
...     print(nums[i])
...
one
two
three
```

On the other hand, the dictionary as a class has the items () method, which creates a special structure consisting of tuples. Each tuple includes a key and a value:

```
>>> n = nums.items()
>>> n
dict_items([(1, 'one'), (2, 'two'), (3, 'three')])
```

In the for loop, you can unpack tuples, thus immediately extracting both the key and its value:

```
>>> for key, value in nums.items():
...     print(key, 'is', value)
...
1 is one
2 is two
3 is three
```

The dictionary methods keys () and values () allow you to obtain separately lists of keys and values. So if, for example, you need to iterate over only the values or only the keys, it is better to use one of these methods:

```
>>> v_nums = []
>>> for v in nums.values():
...     v_nums.append(v)
...
>>> v_nums
['one', 'two', 'three']
```

Dictionary Methods

In addition to the three methods described above, items (), keys (), and values (), dictionaries have eight more. These methods are clear (), copy (), fromkeys (), get (), pop (), popitem (), setdefault (), update ().

The clear () method deletes all elements of the dictionary, but does not delete the dictionary itself. As a result, an empty dictionary remains:

```
>>> a
{'dog': 'собака', 'cat': 'кошка', 'mouse': 'мышь', 'bird': 'птица', 'elephant': 'слон'}
>>> a.clear()
>>> a
{}
```

A dictionary is a mutable data type. Therefore, like a list, it is passed to the function by reference. Therefore, sometimes, in order to avoid undesirable changes in the global dictionary, it is copied. This is done for other purposes.

```
>>> nums2 = nums.copy()
>>> nums2[4] = 'four'
>>> nums
{1: 'one', 2: 'two', 3: 'three'}
>>> nums2
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

The fromkeys () method allows you to create a dictionary from a list whose elements become keys. You can apply the method to both the dict class and its objects:

```
>>> a = [1, 2, 3]
>>> c = dict.fromkeys(a)
>>> c
{1: None, 2: None, 3: None}
>>> d = dict.fromkeys(a, 10)
>>> d
{1: 10, 2: 10, 3: 10}
>>> c
{1: None, 2: None, 3: None}
```

The get () method allows you to get an element by its key:

```
>>> nums.get(1)
'one'
```

Equivalent to nums [1].

The pop () method removes an element from the dictionary by the specified key and returns the value of the deleted pair. The popitem () method takes no arguments, deletes and returns an arbitrary element

```
>>> nums.pop(1)
'one'
```

```
>>> nums
{2: 'two', 3: 'three'}
>>> nums.popitem()
(2, 'two')
>>> nums
{3: 'three'}
```

Using setdefault (), you can add an item to the dictionary:

```
>>> nums.setdefault(4, 'four')
'four'
>>> nums
{3: 'three', 4: 'four'}
```

Equivalent to nums [4] = 'four' if the element with key 4 is not in the dictionary. If it already exists, then nums [4] = 'four' will overwrite the old value, setdefault () will not.

Using update (), you can add another dictionary to the dictionary:

```
>>> nums.update({6: 'six', 7: 'seven'})
>>> nums
{3: 'three', 4: 'four', 6: 'six', 7: 'seven'}
```

The method also updates the values of existing keys. Includes a number of features.

Practical work

1. Create a dictionary by associating it with the school variable and fill it with data that would reflect the number of students in different classes (1a, 1b, 2b, 6a, 7c, etc.). Make changes to the dictionary according to the following: a) in one of the classes the number of students has changed, b) a new class has appeared in the school, c) another class has been disbanded (deleted). Calculate the total number of students in the school.

2. Create a dictionary where the keys are numbers and the values are strings. Apply the items () method to it, transfer the resulting dict_items object to a function you wrote that creates and returns a new dictionary that is "inverse" to the original, that is, the keys are strings, and the values are numbers.

End **39-40 min**	**Reflection**. Pupils analyze activity in the lesson, describe difficulties, suggest ways to overcome them.	

Differentiation – how do you plan to give more support? How do you plan to challenge the more able learners?	**Assessment – how are you planning to check learners' learning?**	**Health and Safety**
Differentiation in the selection of tasks, in the expected result from a particular student, in the provision of individual support to the student at the stage of solving problems.	Mutual evaluation (according to the results of the experiment) Self-assessment (problem solving)	Compliance with safety regulations in the computer science cabinet

Lesson reflection *Were the lesson / learning goals realistic?* *Have all students reached the CO?* *If not, why?* *Is the differentiation done correctly in the lesson?* *Have the temporary stages of the lesson been sustained?* *What deviations were from the lesson plan and why?*	*Use this section to think about the lesson. Answer the most important questions about your lesson from the left column.*