# ЛАБОРАТОРНАЯ РАБОТА №3

## *Тема: ABC-классы. Списки*

## *Цель: Рассмотрение особенностей использования ABC-классов и списков*

### *ОГЛАВЛЕНИЕ*

[***1 ABC-классы***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1)[***1.1 Контейнеры***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.1)[***1.2 Итераторы***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.2)[***1.3 Хэширование объектов***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.3)[***1.4 Последовательности***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4)[***1.4.1 Протокол последовательности***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.1)[***1.4.2 Операции, общие для последовательностей***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)[***1.5 Оператор цикла for in***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.5)[***2 Списки***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2)[***2.1 Создание списков***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2.1)[***2.2 Поддержка списками общих свойств последовательностей***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2.2)[***2.3 Поддержка списками операций, общих для изменяемых последовательностей***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2.3)[***2.4 Использование других методов при работе со списками***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2.4)[***Индивидуальные задания***](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#ind)

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *1 ABC-классы и протоколы*

Помимо типов (классов) данных программы язык Python имеет и так называемые абстрактные базовые классы (англ. abstract base classes) – ABC-классы, каждый из которых объединяет данные нескольких различных типов, имеющих общие свойства. Например, к ABC-классу контейнер (англ. container) относятся все типы данных, которые имеют в своем составе более одного объекта (см. [подраздел 1.1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.1)).  
В табл. 1 приведены названия ABC-классов, классы, их порождающие, и атрибуты.

|  |  |  |
| --- | --- | --- |
| Таблица 1 – ABC-классы | | |
| **ABC-класс** | **Базовые ABC-классы** | **Атрибуты, методы** |
| Container (см. [подраздел 1.1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.1)) |  | \_\_contains\_\_ |
| Hashable (см. [подраздел 1.3](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.3)) |  | \_\_hash\_\_ |
| Iteruble (см. [подраздел 1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.2)) |  | \_\_iter\_\_ |
| Iterator (см. [подраздел 1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.2)) | Iteruble | \_\_next\_\_ |
| Sized |  | \_\_len\_\_ |
| Callable |  | \_\_call\_\_ |
| Sequence (см. [подраздел 1.4](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4)) | Container, Iteruble Sized | \_\_getitem\_\_, \_\_len\_\_ |
| MutableSequence (см. [подраздел 1.4](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4)) | Sequence | \_\_getitem\_\_, \_\_len\_\_, \_\_setitem\_\_, \_\_delitem\_\_ |
| Set(см. [раздел 1](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#1) лаб. раб. №5) |  | \_\_contains\_\_, \_\_iter\_\_, \_\_len\_\_ |
| MutableSet (см. [раздел 1](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#1) лаб. раб. №5) | Set | \_\_contains\_\_, \_\_iter\_\_, \_\_len\_\_, \_\_setitem\_\_, \_\_delitem\_\_ |
| Mapping (см. [раздел 2](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#2) лаб. раб. №5) | Sized, Iterable, Container | \_\_getitem\_\_, \_\_iter\_\_, \_\_len\_\_ |
| MutableMapping (см. [раздел 2](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#2) лаб. раб. №5) | Mapping | \_\_getitem\_\_, \_\_setitem\_\_, \_\_delitem\_\_, \_\_iter\_\_, \_\_len\_\_ |

В отличие от встроенных типов, которые хранятся в модуле *builtins* (см. [подраздел 1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1.2) лаб. раб. №2) и автоматически загружаются в память при вызове интерпретатора, ABC-классы описаны в модуле *collections*. Поэтому для их использования необходимо подключить к программе этот модуль. Начиная с версии Python 3.3 ABC-классы помещены в новый модуль *collections.abc*. При этом с целью обеспечения обратной совместимости сохранена возможность доступа к ABC-классам по-прежнему через модуль *collections*. Использование модуля *collections* и ABC-классов см. в подразделах [1.1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.1), [1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.2), [1.3](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.3), [1.4.1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.1), [2.1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2.1) данной лабораторной работы, в [разделе 2](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#2) и в подразделах [1.1](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.1), [3.3](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.3) лаб. раб. №4, в [разделе 1](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#1) и в подразделах [2.1](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#2.1), [3.1](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#3.1) лаб. раб. №5.  
Модуль *collections* помимо ABC-классов также содержит специализированные типы контейнеров, обеспечивающие альтернативы встроенным типам *list*, *tuple*, *dict* и др. Например, *collections.deque* (очереди), *collections.nametuple* (именнованные кортежи), *collections.OrderedDict* (упорядоченные словари) и др.  
Протоколы в языке Python подобны интерфейсам в других языках в том, что определяют набор методов, который объекты должны поддерживать при использовании протокола, но отличаются тем, что разпознаются на уровне языка и имеют синтаксическую поддержку. Использование протоколов приведено в подразделах [1.1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.1), [1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.2) и [1.4](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4).

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *1.1 Контейнеры*

Контейнеры – это объекты, которые содержат произвольное число других объектов. Они принадлежат ABC-классу *container* (см. табл. 1). Контейнеры делятся на последовательности (см. [подраздел 1.4](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4)), множества (см. [раздел 1](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#1) лаб. раб. №5) и словари (см. [раздел 2](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#2) лаб. раб. №5).  
Все объекты-контейнеры, поддерживая протокол контейнера, имеют специальный метод *\_\_contains\_\_(other)*, который проверяет, содержится ли в данном объекте объект *other*, и возвращает значение True, если содержится, и значение False, если нет.  
Возьмем два объекта: строку "It's a string" и целое число 32, заданные литералами, и попытаемся вызвать для них метод *\_\_contains\_\_()*:  
  
>>> "It's a string" .\_\_contains\_\_( "string" )  
True.  
>>> "It's a string" .\_\_contains\_\_( "cat" )  
False.  
  
Т.е. метод *\_\_contains\_\_()* определил, что подстрока "string" является частью строки "It's a string", а подстрока "cat" – нет. Следовательно, строки являются контейнерами.  
  
>>> (32).\_\_contains\_\_(3)  
AttributeError: 'int' object has no attribute '\_\_contains\_\_' .  
  
Для числа использование метода *\_\_contains\_\_()* привело к возникновению исключения AttributeError (ошибка атрибута), поскольку целочисленный объект не имеет этого метода. Поэтому целые числа не являются контейнерами.  
Отметим, что число 32 взято в скобки для устранения лексической неоднозначности – последовательность символов "32." интерпретатор воспринимает не как целое число, за которым следует оператор точка, а как число с плаващей точкой. Поэтому в этом случае (без использования скобок) будет зафиксирована синтаксическая.ошибка.  
Метод *\_\_contains\_\_()* вызывается при выполнении операторов *in* и *not in* для определения вхождения или невхождения одного объекта в состав другого объекта (см. подразделы [1.4.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2) и [2.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2.2)).  
При создании пользователем контейнера необходимо в объявлении класса описать метод *\_\_contains\_\_()*.  
Для определения, является ли некий объект контейнером или нет, можно также воспользоваться модулем *collections* (см. [раздел 1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1)) для получения ABC-класса *Container* и встроенной функцией *isinstance(object, classinfo)* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#isinstance) лаб. раб. №1), которая возвращает True, если объект, указанный аргументом *object* является экземпляром класса, заданного аргументом *classinfo*. Если *object* не является объектом указанного типа, то функция возвращает значение False.  
Для проверки возьмем те же два объекта: строку и число и, загрузив описание ABC-класса *Container* из модуля *collections:*  
  
>>> from collections import Container,  
  
определим с помощью функции *isinstance()*, принадлежат ли эти объекты классу *container*:  
  
>>> isinstance ( "It's a string" , Container)  
True,  
>>> isinstance (32, Container)  
False.  
  
Новая проверка подтвердила, что строки являются контейнерами, а целые числа (см. [подроздел 4.1.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#4.1.1) лаб. раб. №1) – нет. Также не являются контейнерами числа с пдавающей точкой, комплексные числа и значения логического типа (см. подрозделы [4.1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#4.1.2), [4.1.3](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#4.1.3) и [4.2](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#4.2) лаб. раб. №1).

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *1.2 Итераторы*

Python поддерживает концепцию итерации контейнеров. Ее реализация осуществляется с помощью двух методов, которые позволяют классам, определенным пользователем, поддерживать итерации.  
Первый из этих методов – *\_\_iter\_\_()*, примененный к объекту-контейнеру: *container.\_\_iter\_\_()*, обеспечивает этому объекту поддержку итераций. Метод возвращает объект-итератор (англ. iterator) для объекта-контейнера, что делает его итерабельным (или перечисляемым) объектом (англ. iterable object). Если контейнер поддерживает несколько типов итерации, то для каждого вида итерации должен быть определен свой итератор, например, для объекта структуры дерева, должны быть определены два интератора – один для обхода дерева "сначала вниз", другой – "сначала вправо".  
Итераторы-объекты поддерживают следующие два метода, которые вместе составляют протокол итератора:

* *iterator.\_\_iter\_\_() – возвращает объект-итератор. Это необходимо для того, чтобы итераторы, как и контейнеры, могли быть использованы в операторах for in;*
* *iterator.\_\_next\_\_() – возвращает очередной элемент контейнера. Если элементов больше нет – вырабатывает исключение StopIteration. Python определяет несколько объектов-итераторов для поддержки итераций для общих и специальных типов последовательностей, словарей и других более специализированных форм.*

Проверим, является ли строка "cat" итерабельным объектом:  
  
>>> from collections import Iterable, Iterator  
>>> isinstance ( "cat" , Iterable)  
True.  
>>> isinstance ( "cat" , Iterator)  
False.  
  
Поскольку строки являются итерабельными объектами, но не итераторами, создадим для строки "cat" объект-итератор *i*:  
  
>>> i= 'cat' .\_\_iter\_\_()

и, используя метод \_\_iter\_\_(), получим все символы этой строки:  
  
>>> i.\_\_next\_\_()  
'c'  
>>> i.\_\_next\_\_()  
'a'  
>>> i.\_\_next\_\_()  
't'  
>>> i.\_\_next\_\_()  
StopIteration.  
  
Когда символы строки закончились, вырабатывается исключение *StopIteration*.  
Отметим, что при вызове для объекта-итератора метода *\_\_iter\_\_()* новый объект-итератор не создается, а возвращается ссылка на исходный объект:  
  
>>> id (i)  
33426320  
>>> id (i.\_\_iter\_\_())  
33426320  
  
Вместо метода *\_\_next\_\_()* можно использовать встроенную функцию *next()* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#next) лаб. раб. №1).  
Укажем, что все встроенные типы последовательностей (см. [подраздел 1.4](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4)) являются итерабельными типами.  
Вместо метода *\_\_iter\_\_()* можно использовать встроенную функцию *iter(object[, sentinel])* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#iter) лаб. раб. №1), которая возвращает объект-итератор.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *1.3 Хэширование объектов*

Встроенная функция *hash(object)* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#hash) лаб. раб. №1) возвращает хэш-значение (англ. hash value) объекта (если он его имеет), представленное числом. Хэш-значения используются для ускорения сравнения ключей словарей при их просмотре (см. [раздел 2](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#2) лаб. раб. №5). Если значения объектов численно равны, то хэш-функция возвращает одинаковые значения (даже если объекты принадлежат разным типам). Наример:  
  
>>> hash (1)  
1  
>>> hash (1.0)  
1  
  
Объект является хэшируемым (англ. hashable), если он имеет хэш-значение, которое не меняется в течение срока его существования. Для этого требуется, чтобы для объектп был определен специальный метод *\_\_hash\_\_()*, и чтобы он мог быть сравним с другими объектами, т.е. для него также должен быть определен специальный метод *\_\_eq\_\_()* (см. операции сравнения в [подразделе 6.2](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#6.2) лаб. раб. №1). Хэшируемые объекты принадлежат ABC-классу *Hashable* (см. табл. 1).  
Хэшируемость объекта дает возможность ему быть использованым в качестве ключа словаря (см. [подраздел 2.1](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#2.1) лаб. раб. №5), а также в качестве элементов множеств, поскольку их структуры данных внутренне используют хэш-значения (см. [раздел 1](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#1) лаб. раб. №5).  
Все хэшируемые объекты являются неизменяемыми объектами языка Python. В то время как ни один изменяемый контейнер не является хэшированным объектом.  
Объекты, которые являются экземплярами пользовательских классов, хэшируемы по умолчанию, они при сравнении дают неравентсво и их хэш-значениями являются значения их 4*id()*.  
Специальный метод *\_\_hash\_\_()* вызывается при выполнении встроенной функции *hash()* и при выполнении операций над элементами хэшируемых контейнеров. Метод *\_\_hash\_\_()* должен возвращать число. Единственным условием равенства объектов, является равенство их хэш-значений. Выше установлено, что объекты 1 и 1.0 имеют одинаковые хэш-значения. Поэтому эти объекты равны:  
  
>>> 1==1.0  
True  
  
Отметим, что функция *hash()* обрезает возращаемое значение для методов *\_\_hash\_\_()* пользовательских классов до значения 2\*\*31 (4 байта) на 32-битных и 2\*\*63 (8 байтов) на 64-битных платформах.  
Если класс не определил метод *\_\_eq\_\_()*, он не должен определять и метод *\_\_hash\_\_()*. Если в классе был определен *\_\_eq\_\_()*, но не определен *\_\_hash\_\_()*, его экземпляры не могут использоваться как элементы в хэшируемых коллекциях. Если класс определяет изменяемые объекты и объявляяет метод *\_\_eq\_\_()*, он не должен объявлять метод *\_\_hash\_\_()*, поскольку применение хэшируемых коллекций предусматривает, чтобы их ключи были неизменяемыми объектами.  
Определенные пользователями классы имеют методы *\_\_eq\_\_()* и *\_\_hash\_\_()* по умолчанию. Сравнение со всеми объектами дает неравнество (кроме сравнения объекта с самим собой) и *x.\_\_hash\_\_()* возвращает соответствующее значение, такое что из *x == y* следует, что *x* есть *y* и *hash(x) == hash(y)*.  
Класс, который переопределяет *\_\_eq\_\_()* и не определяет *\_\_hash\_\_()* будет иметь неявно установленное значение *\_\_hash\_\_()*, равное None. Когда метод *\_\_hash\_\_()* класса равен None, экземпляры этого класса вызовут ошибку TypeError при попытке программы получить их хэш-значение, и будут правильно идентифицированы как нехэшируемые объекты при проверке с помощью вызова функции *isinstance(obj, collections.Hashable)*.  
Класс, который переопределяет *\_\_eq\_\_()* и нуждается в применении метода *\_\_hash\_\_()* родительского класса, должен в явном виде указать это интерпретатору, устанавливая *\_\_hash\_\_ = <ParentClass>.\_\_hash\_\_*.  
Класс, который переопределяет *\_\_eq\_\_()* и желает подавить поддержку хэширования, должен включить *\_\_hash\_\_ = None* в описание класса.  
Класс, который так определяет свой собсвтенный метод *\_\_hash\_\_()*, что это вызывет ошибку TypeError, не будет правильно идентифицирован вызовом функции *isinstance(obj, collections.Hashable)*.  
Рассмотрим теперь взаимосвязь между понятиями кэшируемости объектов и их изменяемости. В языке Python все объекты, которые имеют хэш, являются неизменяемыми и ни один изменяемый объект не имеет хэш. Это означает, что из хэшируемости объекта следует его неизменяемость, а из изменяемости следует его нехэшируемость. Но эти понятия не равнозначны, поскольку из неизменяемости объъекта не следует его хэшируемость.  
Для примера определим, изменяемыми или неизменяемыми являются такие объекты как строки и числа с плавающей точкой. Для этого воспользуемся *ABC-классом* (см. [раздел 1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1)) и встроенной функцией *isinstance(object, classinfo)* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#isinstance) лаб. раб. №1):  
  
>>> from collections import Hashable  
>>> isinstance ( 'cat' , Hashable)  
True.  
>>> isinstance (0.25, Hashable)  
True.  
  
Т.е. строки и числа сплавающей точкой являются хэшируемыми, а, следовательно, неизменяемыми объектами (так же, как целые числа, комплекстные числа и данные логического типа).

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *1.4 Последовательности*

Последовательностями, т.е. объектами, относящимися к ABC-классу *Sequence*, в языке Python являются контейнеры (см. [подраздел 1.1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.1)), которые поддерживают протокол последовательности (см. [подраздел 1.4.1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.1)).

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *1.4.1 Протокол последовательности*

Чтобы объекты могли поддерживать протокол последовательности, требуется чтобы для них были определены следующие два специальных метода:

* *\_\_len\_\_() – возвращает число элементов последовательности (n);*
* *\_\_getitem\_\_(key) – возвращает значение элемента последовательности для данного ключа key. Значение ключа должно быть положительным целым числом в диапазоне от 0 до n-1 или отрицательным в диапазоне от -1 до -n.*

Проверим, определены ли эти методы для строк и целых чисел:  
  
>>> 'Cat' .\_\_len\_\_()  
3  
>>> (125).\_\_len\_\_()  
AttributeError: 'int' object has no attribute '\_\_len\_\_'  
>>> 'Cat' .\_\_getitem\_\_(1)  
'a'  
  
Результаты показывают, что строки поддерживают оба метода протокола и поэтому являются последовательностями. А целые числа не поддерживают уже первый метод (второй – тоже) и, следовательно, не принадлежат классу *Sequence*. Отметим, что доступ к значениям элементов последовательности по их номерам означает упорядоченность значений последовательностей.  
  
Чтобы объекты относилисть к ABC-классу *Mutable Sequence* – изменяемой последовательности, необходимо, чтобы для них были определены дополнительно еще два специальных метода:

* *\_\_setitem\_\_(key, value) – добавляет в последовательность значение value, соответствующее ключу key;*
* *\_\_delitem\_\_(key) – удаляет из последовательности элемент, соответствующий ключу key.*

Проверим, являются ли строки изменяемой последовательностью. Попробуем изменить второй символ строки:  
  
>>> 'Cat' .\_\_setitem\_\_(1, 'u' )  
AttributeError: 'str' object has no attribute '\_\_setitem\_\_'  
  
Т.е., строки не имеют атрибута *\_\_setitem\_\_* и, следовательнло, не принадлежат к классу изменяемых последовательностей. Это неизменяемые объекты.  
К четырем основным правилам предъявляются такде такие требования:

* *если ключ в последовательности является отрицательным числом, то отчет индексов элементов последовательности начинается с конца;*
* *ключи дожны быть соответствующего тииа, иначе возникает исключение TypeError;*
* *если индекс последовательности правильного типа, но превосходит разрешенный диапазон, возникает исключение IndexError.*

Осуществить проверку, является ли некоторый объект неизменяемой последовательностью, можно и по-другому, не привлекая протокол последовательности, а используя встроенную функцию *isinstance()* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#isinstance) лаб. раб. №1) и *ABC-классы*: *Sequence* и *MutableSequence* (см. [раздел 1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1)):  
  
>>> from collections import Sequence, MutableSequence  
>>> isinstance ( 'cat' , Sequence)  
True.  
>>> isinstance ( 'cat' , MutableSequence)  
False.  
  
Т.е. получаем тот же результат – строки действительно являются неизменяемыми последовательностями.  
К базовым последовательностям относяться следующие типы данных:

* *списки (см.*[*раздел 2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2)*);*
* *кортежи (см.*[*раздел 1*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1)*лаб. раб. №4);*
* *диапазоны (см.*[*раздел 2*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#2)*лаб. раб. №4).*

К дополнительным последовательностям относятся следующие типы данных:

* *текстовые последовательности (строки) (см.*[*раздел 2*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2)*лаб. раб. №2);*
* *двоичные последовательности (см.*[*раздел 3*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3)*лаб. раб. №4).*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *1.4.2 Операции, общие для последовательностей*

Все указанные выше типы последовательностей относятся к итерабельным типам данных (см. [подраздел 1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.2)). Ниже приводится список операций, которые являются общими для этих последовательностей. Операции для последовательностей приводятся в порядке возрастания приоритета (в скобках указаны соответствующие им специальные методы). *S* и *T* обозначают последовательности одного типа, *x* – элемент последовательности, *i*, *j*, *k* и *n* – целые числа.  
Для каждой операции указана ссылка на соответствующую ей операцию над строками, которые были рассмотрены в [разделе 2](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2)лаб. раб №2:

* *x in S – равно True, если элемент из S равен x, иначе False (S.\_\_contains\_\_(x)) (см.*[*раздел 2.3*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.3)*лаб. раб. №2);*
* *x not in S – равно False, если элемент из S равен x, иначе True (см.*[*раздел 2.3*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.3)*лаб. раб. №2);*
* *S + T – выполняется операция конкатенации между S и T (S.\_\_add\_\_(T)) (см.*[*раздел 2.3*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.3)*лаб. раб. №2);*
* *S \* n or n \* S – выполняется n раз копирование и конкатенация S (S.\_\_mul\_\_(n));*
* *S[i] – берется i-й элемент S (см.*[*раздел 2.4*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.4)*лаб. раб. №2) (S.\_\_getitem(i));*
* *S[i:j] – срез S с i-го элемента по j-й (см.*[*раздел 2.5*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.5)*лаб. раб. №2);*
* *S[i:j:k] – срез S с i-го элемента по j-й с шагом k (см.*[*раздел 2.5*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.5)*лаб. раб. №2);*
* *встроенная функцмя len(container) – возвращает длину (количество элементов) для контейнера (см.*[*подраздел 1.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.1)*), эквивалентна методу S.\_\_len\_\_() (см.*[*раздел 2.6*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.6)*лаб. раб. №2);*
* *встроенная функцмя min(iteruble[,keyword\_only]) – если итерабельный объект iteruble не пуст, возвращает его минимальный элемент. В противном случае – возвращает пустой итерабельный объект keyword\_only (см.*[*раздел 2.6*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.6)*лаб. раб. №2);*
* *встроенная функцмя max(iteruble[,keyword\_only]) – если итерабельный объект iteruble не пуст, возвращает его максимальный элемент. В противном случае – возвращает значение keyword\_only (см.*[*раздел 2.6*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.6)*лаб. раб. №2);*
* *метод s.index(x[, i[, j]]) – возвращает индекс первого вхождения x в s (начиная с индекса i и до индекса j) (см.*[*подраздел 2.7*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.7)*лаб. раб. №2):  
    
  >>> '1020304050' .index( '0' ,2)  
  3;*
* *метод s.count(x) – возвращает число вхождений x в s (см.*[*подраздел 2.7*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.7)*лаб. раб. №2):  
    
  >>> '1020304050' .count( '0' )  
  5.*

Отметим, что операции *in* и *not in* имеют тот же приоритет, что и операции сравнения. Операции "+" и "\*" имеют такой же приоритет, как и соответствующие операции над числами.  
Последовательности одинаковых типов также поддерживают операции сравнения. Две последовательности будут равны, если имеют одинаковый тип, равное число элементов и все элементы одной последовательности равны соответствующим элементам другой последовательности.  
Единственной операцией, которая поддерживается неизменяемыми последовательностями и не поддерживается изменяемыми последовательностями, является вызов встроенной функции *hash()* (см. [подраздел 1.3](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.3)). Эта поддержка позволяет использовать неизменяемые последовательности, такие как кортежи (см. [разделе 1](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1) лаб. раб. №4), в качестве ключей словарей (см. [подраздел 2.1](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#2.1) лаб. раб. №5) и сохранять их в экземплярах объектов *set* и *frozenset* (см. [раздел 1](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#1) лаб. раб. №5).  
Помимо применения последовательностей встроенных типов пользователь может, используя выше приведенные методы правил, создавать собственные типы данных, являющиеся последовательностями.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *1.5 Оператор цикла for in*

Оператор цикла *for in* представляет собой цикл обхода заданной последовательности элементов, которая должна быть итерабельным объектом, и выполнения в своем теле различных операций над ними. Имеет следующую форму:  
  
for <элемент> in <последовательность элементов>:  
<блок> .  
При выполнении оператора *for in* интерпертатор осуществляет следующие действия. Сначала для итерабельного объекта вызывает метод *\_\_iter\_\_*, чтобы получить его итератор. Затем последовательно для итератора вызывается метод *\_\_next\_\_*, который возвращает очередной элемент последовательности, до тех пор, пока не возникнет исключение *StopIteration*. Эти действия соответствуют протоколу итераций (см. [подраздел 1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.2)).  
Поскольку строка представляет собой последовательность и, следовательно, является итерабельным типом (см. [подраздел 1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.2)), то можно использовать оператор *for in* для перебора всех символов строки и вывода их с дополнительным пробелом:  
  
>>> s= 'Микропроцессор'  
>>> for l in s:  
print (l,end= ' ' )  
М и к р о п р о ц е с с о р.  
  
По умолчанию функция *print()* после вывода сообщения на экран осуществляет перевод строки. Однако с помощью параметра *end=<символ>*можно указать символ, которым будет заканчиваться вывод. В данном случае это пробел.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *2 Списки*

Списки в языке Python имеют тип (класс) *list* и представляют собой последовательности элементов, значениями которых, в отличие от строк, могут быть данные разных типов, в том числе и списки. Другим отличием списков от строк является то, что они являются изменяемыми объектами (см. [раздел 2](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2) лаб. раб. №2), т.е. значения элементов списка можно изменять без создания нового списка. Списки в языке Python похожи на индексные массивы языка Jaxascript.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *2.1 Создание списков*

Существует два подхода для создания списка:

* *первый подход заключается в использовании квадратных скобок для задания литерала списка:*
  + *литерал может содержать элементы списка, разделенные запятыми:  
      
    >>> a\_list=[0.01, 'Web' , [1,2,3]];  
    >>> a\_list  
    [0.01, 'Web', [1, 2, 3]];*
  + *литерал может содержать генератор элементов списка (см.*[*подраздел 3.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#3.1)*лаб. раб. №5);*
  + *если литерал ничего не содержит – создается пустой список:  
      
    >>> a\_list=[]  
    >>> a\_list  
    [];*
* *второй подход предусматривает использование конструктора – встроенной функции list([iterable]) (см.*[*подраздел 8.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#list)*лаб. раб. №1), аргумент которой – итерабельный тип данных преобразуется в список:  
    
  >>> b\_list= list ( 'Python' )  
  >>> b\_list  
  ['P', 'y', 't','h','o','n'].  
  >>> type (b\_list)  
  <'class 'list'>.  
    
  Если аргументом конструктора является объект типа list, то создается копия списка, если аргумент не указан – создается пустой список.*

В логическом контексте пустой список (*[]*) соответствует False, любой другой – True.  
Проверим, являются ли списки итерабельными объектами (см. [подраздел 1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.2)):  
  
>>> from collections import Iterable  
>>> isinstance ([1,2,3], Iterable)  
True.  
Все итерабельные объекты поддерживают следующие встроенные функции:

* *all([iterable]) – возвращает True, если все значения итерабельного объекта имеют значение True или если итерабельный объект пуст:  
    
  >>> all ([1,2,3])  
  True,  
  >>> all ([])  
  True,  
  >>> all ([1,2,0])  
  False;*
* *any([iterable]) – возвращает True, если хотя бы одно значение итерабельного объекта равно True. Если итерабельный объект пуст, возвращает False:  
    
  >>> any ([1,2,0])  
  True,  
  >>> any ([])  
  False,  
  >>> any ([0,0,0])  
  False;*

Теперь проверим, являются ли списки изменяемыми последовательностями (см. также проверку для строк в [подразделе 1.4.1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.1)):  
  
>>> from collections import Sequence, MutableSequence  
>>> isinstance (a\_list, Sequence)  
True,  
>>> isinstance (a\_list, MutableSequence)  
True.  
  
Действительно, списки являются изменяемыми последовательностями.

A CLASS="out" HREF="#beg" style="margin-left:80%">Оглавление

#### *2.2 Поддержка списками операций, общих для последовательностей*

Будучи последовательностями, списки поддерживают все операции, которые являются общими для всех типов последовательностей (см. [подраздел 1.4.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)):

* *x in s – равно True, если элемент из s равен x, иначе False. В списках используется для проверки вхождения элемента в список:  
    
  >>> 'Web' in a\_list  
  True  
  >>> 1 in a\_list  
  False;*
* *x not in s – равно False, если элемент из s равен x, иначе True;*
* *s + t – выполняется операция конкатенации между s и t (объединение списков):  
    
  >>> id (a\_list)  
  36877496  
  >>> a\_list+=[0x25]  
  >>> a\_list  
  [0.01, 'Web', [1, 2, 3], 37]  
  >>> id (a\_list)  
  36877496  
    
  Видно, что для изменяемых типов операция конкатенации не создает нового объекта, поскольку значение функции id(), т.е. адрес объекта, остается неизменным. Меняется только его значение. В отличие от неизменяемого типа, для которого операция конкатенации создает новый объект с другим адресом:  
    
  >>> s= 'cat'  
  >>> id (s)  
  35173984  
  >>> s+= 's'  
  >>> s  
  'cats'  
  >>> id (s)  
  37084672;*
* *s \* n or n \* s – выполняется n раз копирование ссылок элементов s:  
    
  >>> [1,2,3]\*2  
  [1, 2, 3, 1, 2, 3];*
* *s[i] выбирается i-й элемент s. Индексация списков похожа на индексацию строк – индексы могут быть как положительными, так и отрицательными числами:  
    
  >>> a\_list[1]  
  'Web'  
  >>> a\_list[-2]  
  [1, 2, 3].  
  Для доступа к элементам списка, которые сами являются списками, необходимо указывать в квадратных скобках два индекса – первый для элементов основного списка, второй – для элементов внутреннего списка. Степень вложенности списков не ограничена:  
    
  >>> a\_list[2][0]  
  1;*
* *s[i:j] – срез s с i-го элемента по j-й:  
    
  >>> a\_list[:2]  
  [0.01, 'Web']  
  >>> a\_list[::-1]  
  [37,[1, 2, 3], 'Web', 0.01];*
* *s[i:j:k] – срез s с i-го элемента по j-й с шагом k:  
    
  >>> a\_list[1:4:2]  
  ['Web', 37];*
* *встроенная функцмя len(s) (см.*[*подраздел 8.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#len)*лаб. раб. №1) – возвращает длину (количество элементов) для аргумента s, который может быть последовательностью любого типа, множеством (см.*[*раздел 1*](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#1)*лаб. раб. №5) или словарем (см.*[*раздел 2*](https://py-khpi.github.io/!nfo/doc/lab/LAB5/lab.htm#2)*лаб. раб. №5):  
    
  >>> len (a\_list)  
  4;*
* *встроенная функцмя min(s) (см.*[*подраздел 8.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#min)*лаб. раб. №1) – возвращает минимальный элемент s. Для списков необходимо, чтобы их элементы были сравнимы между собой, т.е. все элементы должны быть или числами, или строками, или списками:  
    
  >>> min ([4,2,7])  
  2  
  >>> min ([ 'cat' , 'www' ])  
  'cat'  
  >>> min ([[11,2],[10,7],[26,0]])  
  [10, 7]  
  >>> min ([ 'cat' ,77])  
  TypeError: unorderable types: int() > str();*
* *встроенная функцмя max(s) (см.*[*подраздел 8.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#max)*лаб. раб. №1) – возвращает максимальный элемент s:  
    
  >>> max ([1,9],[3,1])  
  [3, 1];*
* *метод s.index(x[, i[, j]]) – возвращает индекс первого вхождения x в s (начиная с индекса i и до индекса j):  
    
  >>> [0,1,1,0,2,2,0,3,3,0,4,4].index(0,4)  
  6.  
    
  Если искомого элемента нет в списке:  
    
  [0,1,1,0,2,2,0,3,3,0,4,4].index(5),  
    
  выводится сообщение об ошибке:  
    
  ValueError: 5 is not in list,  
    
  т.е. элемент 5 не найден в списке;*
* *метод s.count(x) – возвращает число вхождений x в s:  
    
  >>> [0,1,1,0,2,2,0,3,3,0,4,4].count(0)  
  4*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *2.3 Поддержка списками операций, общих для изменяемых последовательностей*

Поскольку списки являются изменяемыми последовательностями (см. [подраздел 2.1](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2.1)), они поддерживают все операции, которые являются общими для любой изменяемой последовательности *s*:

* *s[i] = x – i-й элемент последовательности s принимает значение x:  
    
  >>> a\_list[3]= '37'  
  >>> a\_list  
  [0.01, 'Web', [1, 2, 3], '37'];*
* *s[i:j] = t – срез последовательности s от i до j заменяется содержимым объекта итерабельного типа t (t должен иметь ту же длину, что и срез):  
    
  >>> a\_list[1:3]= 'ab'  
  >>> a\_list  
  [0.01, 'a', 'b', '37'];*
* *del s[i:j] – то же самое, что s[i:j] = [], т.е. удаление элементов среза:  
    
  >>> del a\_list[1:3]  
  >>> a\_list  
  [0.01, '37'];*
* *s[i:j:k] = t – элементы среза s[i:j:k] заменяются значениями итерабельного типа t;*
* *del s[i:j:k] – элементы среза s[i:j:k] удаляются из списка;*
* *s.append(x) – добавляет элемент x к концу последовательности (то же самое, что и s[len(s):len(s)] = [x]):  
    
  >>> a\_list.append( True )  
  >>> a\_list  
  [0.01, '37', True];*
* *s.clear() – удаляет все элементы из последовательности s (то же самое, что и del s[:]):  
    
  >>> b\_list=[10,20,30]  
  >>> b\_list.clear()  
  >>> b\_list  
  [];*
* *s.copy() – создает "поверхностную" (англ. shallow) копию s (то же самое, что и s[:]):  
    
  >>> b\_list=a\_list.copy()  
  >>> b\_list  
  [0.01, '37', True];*
* *s.extend(t) – расширяет s содержимым объекта итерабельного типа t (то же самое, что и s[len(s):len(s)] = t):  
    
  >>> a\_list.extend([ 'X' , 'Y' ])  
  >>> a\_list  
  [0.01, '37', True, 'X', 'Y'];*
* *s.insert(i, x) – вставляет x в последовательность s на место элемента с индексом i (то же самое, что и s[i:i] = [x])  
    
  >>> a\_list.insert(2,[1,3,2])  
  >>> a\_list  
  [0.01, '37', [1, 3, 2], True, 'X', 'Y'];*
* *s.pop([i]) – возвращает значение элемента с индексом i, а сам элемент удаляется из последовальности s:  
    
  >>> a\_list.pop(1)  
  '37'  
  >>> a\_list  
  [0.01, [1, 3, 2], True, 'X', 'Y'],  
    
  по умолчанию значение индекса i равно -1, поэтому если индекс не указан, то возвращается значение и и происходит удаление последнего элемента последоваетельности:  
    
  >>> a\_list.pop()  
  'Y'  
  >>> a\_list  
  [0.01, [1, 3, 2], True, 'X']*
* *s.remove(x) – удаляет первый элемент последовательности s, который равен x (s[i] == x):  
    
  >>> a\_list.remove(0.01)  
  >>> a\_list  
  [[1, 3, 2], True, 'X'],  
    
  если x не найден – возникает исключение ValueError:  
    
  >>> a\_list.remove(0.01)  
  ValueError: list.remove(x): x not in list;*
* *s.reverse() – переставляет элементы последовательности в обратном порядке:  
    
  >>> a\_list.reverse()  
  ['X', True, [1, 3, 2]]  
    
  Эту же операцию можно выполнить, используя срез списка:  
    
  >>> a\_list[::-1]  
  [[1, 3, 2], True, 'X']*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *2.4 Использование других методов при работе со списками*

Объекты класса *list*, кроме описанных выше, имеют также метод *list.sort()*, который выполняет сортировку элементов списка (будет рассмотрен в [подразделе 1.4](https://py-khpi.github.io/!nfo/doc/lab/LAB6/lab.htm#1.4) лаб. раб. №6, поскольку в общем случае при его использовании требуется знать, как создаются пользовательские функции).

С работой со списками связаны и два метода класса *str*:

* *s.split([sep[,index]]) – возвращает список строк, используя разделитель sep, если аргумент sep не задан, в качестве разделителя применяется пробел. Второй необязательный аргумент указывает на индекс последнего элемента списка, если не указан – без ограничений (см. также*[*подраздел 2.6*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.6)*лаб. раб. №2):  
    
  >>> s= 'cat and dog'  
  >>> s.split()  
  ['cat', 'and', 'dog']  
  >>> s.split( ' ' ,1)  
  ['cat', 'and dog']*
* *s.join(seq) – возвращает строку, состоящую из элементов последовательности (списка), указанной в качестве аргумента. Разделителем этой строки, является значение строки, которая применяет этот метод (см. также*[*подраздел 2.6*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.6)*лаб. раб. №2  
    
  >>> '-' .join([ '1' , '2' , '3' ])  
  '1-2-3'*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg)

#### *Индивидуальные задания*

Разработать программу на языке Python, которая выполняет следующее:

* *1 Создает список a\_list, элементами которого являются объекты типов: int, float, bool, strl и list (элементы задаются в любом порядке).*
* *2 Для каждого элемента списка a\_list определяет:*
  + *2.1 тип элемента;*
  + *2.2 ABC-класс (если элемент принадлежит к некоторому ABC-классу).*
* *3 Для списка a\_list выполняет операции, номера которых заданы в колонке "Операции" табл. №2:*
  + *1 – s[i]=x;*
  + *2 – s[i:j]=t;*
  + *3 – s.append(x);*
  + *4 – s.extend(t);*
  + *5 – s.insert(i,x);*
  + *6 – s.pop([i]);*
  + *7 – s.remove(x);*
  + *8 – s.reverce().*
* *4 Создает строку s, содержащую слова или числа (см. колонку "Cтрока"/"Содержимое" табл. №2), разделенные символами, указанными в колонке "Cтрока"/"Разделитель" табл. №2:*
  + *1 – знак подчеркивания ("\_");*
  + *2 – дефис ("-");*
  + *3 – символ " | ";*
  + *4 – символ "&";*
  + *5 – символ "#".*
* *5 Преобразует строку s в список b\_list, элементами которого являются строки или числа (согласно колонки "Cтрока"/"Содержимое" табл. №2).*
* *6 Выполняет над элементами списка b\_list функции, указанные в колонке "Функции":*
  + *1 – len(), min();*
  + *2 – len(), max();*
  + *3 – min(), max().*
* *7 Выводит результаты выполнения функций на экран.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 2 – Перечень индивидуальных заданий | | | | |
| **Номер п/п** | **Операции** | **Строка** | | **Фукции** |
| **Содержимое** | **Разделитель** |
| 1 | 1,3,6 | слова | 1 | 1 |
| 2 | 2,4,5 | числа | 2 | 2 |
| 3 | 3,7,8 | слова | 3 | 3 |
| 4 | 4,6,1 | числа | 4 | 1 |
| 5 | 5,2,8 | слова | 5 | 2 |
| 6 | 7,2,1 | числа | 1 | 3 |
| 7 | 3,8,4 | слова | 2 | 1 |
| 8 | 1,3,6 | числа | 3 | 2 |
| 9 | 2,4,5 | слова | 4 | 3 |
| 10 | 3,7,8 | числа | 5 | 1 |
| 11 | 4,6,1 | слова | 1 | 2 |
| 12 | 5,2,8 | числа | 2 | 3 |
| 13 | 7,2,1 | слова | 3 | 1 |
| 14 | 3,8,4 | числа | 4 | 2 |
| 15 | 1,3,6 | слова | 5 | 3 |
| 16 | 2,4,5 | числа | 1 | 1 |
| 17 | 3,7,8 | слова | 2 | 2 |
| 18 | 4,6,1 | числа | 3 | 3 |
| 19 | 5,2,8 | слова | 4 | 1 |
| 20 | 7,2,1 | числа | 5 | 2 |

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#beg) `