# ЛАБОРАТОРНАЯ РАБОТА №4

## *Тема: Кортежи. Диапазоны. Двоичные последовательности*

## *Цель: Изучение способов работы с кортежами, диапазонами и двоичными последовательностями*

### *ОГЛАВЛЕНИЕ*

[***1 Кортежи***](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1)[***1.1 Создание кортежей***](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.1)[***1.2 Модуль random***](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.2)[***1.3 Создание игры "Анаграмма"***](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.3)[***2 Диапазоны***](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#2)[***3 Двоичные последовательности***](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3)[***3.1 Тип bytes***](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.1)[***3.2 Тип bytearray***](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.2)[***3.3 Операции над объектами типов bytes и bytearray***](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.3)[***Индивидуальные задания***](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#ind)

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)

#### *1 Кортежи*

Кортежи в языке Python имеют тип (класс) *tuple* и представляют собой неизменяемые последовательности элементов, значениями которых могут быть данные разных типов. По сути это списки, которые нельзя после создания изменять.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)

#### *1.1 Создание кортежей*

Подобно другим рассмотренным типам кортежи могут быть созданы как в виде литералов, так и с помощью консруктора класса – встроенной функции *tuple([iterable])* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#tuple) лаб. раб. №1).  
При использовании литерала в круглые скобки помещают элементы кортежа, разделенные запятыми:  
  
>>> a\_tuple=(3.45, 'JS' , [1,2])  
>>> a\_tuple  
(3.45, 'JS', [1, 2]).  
  
Можно круглые скобки не указывать, перечисляя элементы кортежа через запятые. Результат будет тот же:  
  
>>> a\_tuple=3.45, 'JS' , [1,2]  
>>> a\_tuple  
(3.45, 'JS', [1, 2]).  
  
Выполняется и обратная операция – отдельным объектам присваиваются значения элементов кортежа:  
  
>>> a,b,c=a\_tuple  
>>> a  
3.45  
>>> b  
'JS'  
>>> c  
[1, 2].  
  
При использовании функции *tuple([iterable])* в качестве ее аргумента указывается итерабельный тип, из элементов которого будет создан кортеж:  
  
>>> a\_list=[0.01, 'Web' , 37, True ]  
>>> b\_tuple= tuple (a\_list)  
>>> b\_tuple  
(0.01, 'Web', 37, True).  
  
При вызове конструктора без аргумента:  
  
>>> tuple ()  
()  
  
создается пустой кортеж – *()*, который в логическом контексте соответствует False (любой другой кортеж – True).  
Поскольку кортежи являются неизменяемыми последовательностями:  
  
>>> from collections import Sequence, MutableSequence  
>>> isinstance (a\_tuple, Sequence)  
True,  
>>> isinstance (a\_tuple, MutableSequence)  
False,  
  
они поддерживают все операции, свойственные этим типам объектов (см [подраздел 1.4.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2) лаб. раб. №3). Например, операцию добавления элемента к кортежу:  
  
>>> a\_tuple+= 'a' , 'b'  
(3.45, 'JS', [1, 2], 'a', 'b').  
  
Однако при добавлении одноэлементного кортежа:  
  
>>> a\_tuple+=(77)  
  
возникает ошибка:  
  
TypeError: can only concatenate tuple (not "int") to tuple,  
  
т.е. присоединить к кортежу можно только кортеж (не целое число). Дело в том, что (77) было воспринято не как одноэлементный кортеж, а как выражение целого типа. Поэтому одноэлементный кортеж необходимо задать следующим образом:  
  
>>> a\_tuple+=(77,)  
>>> a\_tuple  
(3.45, 'JS', [1, 2], 'a', 'b', 77).  
  
Работа с кортежами выполняется быстрее, чем со списками. Поэтому, если имеется набор значений, который не будет меняться, его лучше реализовать как кортеж. Кортежи делают код безопаснее в том случае, если нужно защитить данные от записи.  
Примером использования кортежей служит разработка игры "Анаграмма", рассмотренная в [подразделе 1.3](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.3). Поскольку при этом возникает нобходимость получения случайных величин, рассмотрим методы их генерации, представленные в модуле [random](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.2).

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)

#### *1.2 Модуль random*

Загрузим модуль *random* и определим с помощью встроенной функции *dir()* (см. подразделы [8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#dir) лаб. раб. №1 и [2.4](https://py-khpi.github.io/!nfo/doc/lab/LAB6/lab.htm#2.4) лаб. раб. №6 ) пространство имен этого модуля:  
  
>>> import random  
>>> dir (random)  
  
['BPF', 'LOG4', 'NV\_MAGICCONST', 'RECIP\_BPF', 'Random', 'SG\_MAGICCONST', 'SystemRandom', 'TWOPI',  
'\_BuiltinMethodType', '\_MethodType', '\_Sequence', '\_Set', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_',  
'\_\_doc\_\_', '\_\_file\_\_', '\_\_initializing\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_acos',  
'\_ceil', '\_cos', '\_e', '\_exp', '\_inst', '\_log', '\_pi', '\_random', '\_sha512', '\_sin', '\_sqrt',  
'\_test', '\_test\_generator', '\_urandom', '\_warn', 'betavariate', 'choice', 'expovariate',  
'gammavariate', 'gauss', 'getrandbits', 'getstate', 'lognormvariate', 'normalvariate',  
'paretovariate', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangular', 'uniform', 'vonmisesvariate', 'weibullvariate'];  
Как видно из приведенного выше списка имен, модуль *random* содержит большое количество методов, связанных с генерацией случайных чисел (точнее, псевдослучайных чисел). Из них рассмотрим следующие:

* *seed(a=None, version=2) – выполняет инициализацию генератора псевдослучайных чисел. Если параметр a опущен (значение по умолчанию) – используется текущее системное время, иначе используются средства операционной системы (см. описание модуля ssl (Secure Sockets Layer)). Если указана версия 2 (значение по умолчанию) – объекты типов str (см.*[*раздел 2*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2)*лаб. раб. №2), bytes или bytearray (см. подразделы*[*3.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.1)*и*[*3.2)*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.2)*преобразуются в тип int с использованием всех битов, иначе используется встроенная функция hash() (см.*[*подраздел 8.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#hash)*лаб. работы №1);*
* *randrange ([start,] stop [,step] ) – возвращает целое число r, выбранное случайным образом из последовательности целых чисел, созданных встроенной функцией range([start,] stop [, step]) (см.*[*подраздел 8.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#range)*лаб. раб. №1):*
  + *если задан только один обязательный аргумент stop – в диапазоне 0 <= r < stop;*
  + *если заданы два аргумента start и stop – в диапазоне start <= r < stop;*
  + *если заданы все три аргумента – в диапазоне start <= r < stop с шагом step, т.е. r должно быть равно start+step\*n (где n=0,1,..),*

*например, после инициализации генератора случайных чисел:  
  
>>> random.seed()  
  
и восьми вызовов метода random.randrange:  
  
>>> random.randrange(1,20,3)  
  
были получены такие числа: 1, 7, 1, 16, 7, 4, 1, 19;*

* *randint(a, b) – возвращает случайное целое число r, такое, что a <= r <= b. Эквивалентно вызову метода randrange(a, b+1);*
* *choice (seq) – возвращает случайным образом один элемент из непустой последовательности seq, например, при последовательном вызове метода random.choice:  
    
  >>> random.choice([ 'красный' , 'зеленый' , 'синий' ])  
    
  были получены такие значения: 'зеленый', 'красный', 'синий', 'синий', 'зеленый' и т.д.  
  Ели задана пустая последовательность – вырабатывает исключение IndexError;*
* *random() – возвращает случайное число с плавающей запятой r в диапазоне [0.0, 1.0), т.е. 0.0 <= r <1.0, например, при двух вызовах метода:  
    
  >>> random.random()  
    
  возвращаются числа: 0.6147093926689667, 0.00874292814552824;*
* *uniform (a ,b) возвращает случайное число с плавающей запятой r, такое, что a <= r <= b для a <= b и b <= r <= a для b < a. Конечное значение b может быть включено, а может быть и не включено в последовательность – это зависит от округления числа с плавающей точкой, равного a + (b-a) \* random(). Например, при двух вызовах метода:  
    
  >>> random.uniform (10 , 20)  
    
  возвращаются числа: 10.875155265144365, 16.20489723889726.*

Необходимо отметить, что модуль *random* не должен быть использован в приложениях, где требуется криптографическая безопастность генератора псевдослучайных чисел. Вместо него применяйте метод *ssl.RAND\_bytes()* модуля *ssl*.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)

#### *1.3 Создание игры "Анаграмма"*

Рассмотрим программу *ana.py*, которая предназначена для создания игры "Анаграмма" (анаграммы – это слова, образованные из исходного слова путем перестановки букв). Суть игры заключается в следующем: программа выбирает слово (из списка заранее приготовленных), переставляет в нем случайным образом буквы, формируя анаграмму, и предлагает ее пользователям. Их задачей является отгадать исходное слово:  
  
# Создание анаграмм  
import random  
WORDS= 'компьютер' ,  
'питон' ,  
'сервер' ,  
'клиент' ,  
'браузер' ,  
'протокол' ,  
'программа' ,  
'процессор' ,  
'контекст'  
word=random.choice(WORDS)  
correct=word  
ana= ''  
while word:  
pos=random.randrange( len (word))  
ana+=word[pos]  
word=word[:pos]+word[pos+1:]  
print ( '''  
<<Игра АНАГРАММА>>  
Для выхода нажмите Enter ''' )  
print ( 'Вот анаграмма: ' ,ana.upper())  
ans= input ( 'Попробуй отгадать слово ' )  
while ans != correct and ans != '' :  
print 'Ответ неправильный' )  
ans= input ( 'Попробуй еще раз ' )  
if ans == correct:  
print 'Молодец!' )  
print 'Спасибо за игру' )  
input ( ).  
  
Поскольку при создании анаграмм выбор слова и перестановку букв в нем необходимо осуществлять случайным образом, программа начинается с подключения модуля [random](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.2) (см. подраздел 1.2), который содержит необходимые методы для выполнения случайного выбора.  
Набор исходных слов, из которых будут формироваться анаграммы задается в виде кортежа, имя которого (WORDS) задано заглавными буквами. Это сделано для того, чтобы показать, что его значениями являются константы. Использование констант делает программу более понятной и легко читаемой.  
Необходимо однако сказать, что правило задавать константы заглавными буквами в языке Python является прежде всего соглашением между программистами и интерпретатором не поддерживается, т.е. значения, имена которых заданы заглавными буквами можно изменить точно так, как значения, имена которых заданы строчными буквами. Но в силу указанных выше причин этого делать не следует.  
Отметим, что в программе элементы кортежа при перечислении указываются с новой строки без использования символов обратного слэша, как это предусмотрено при задании длинных операторов (см. [подраздел 6.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#6) лаб. раб. №1) и длинных строк (см. [подраздел 2.1](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.1) лаб. раб. №2). Это допускается делать при перечислении элементов последовательностей, указанных в круглых "()", квадратных "[]" и фигурных "{}" скобках.  
Для выбора одного из слов кортежа WORDS используется метод *choice()* модуля *random*(см. [подраздел 1.2](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.2), который случайным образом выбирает один из элементов кортежа и сохраняет его в переменной *word*.  
Затем начинается собственно процесс формирования анаграммы в переменной *ana*, которая вначале является пустой строкой. При этом в цикле, пока переменная *word* не примет значение *''*, что в логическом контексте означает *False*, будут выполняться следующие действия:

* *с использованием метода randrange() модуля random (см.*[*подраздел 1.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.2)*) случайным образом выбирается номер буквы слова word (диапазон случайных чисел задан от 0 до значения, которое на единицу меньше числа букв слова, что позволяет выбрать любую букву слова – от первой до последней);*
* *формируется новое значение переменной ana, к которому добавляется выбранная буква слова word;*
* *формируется новое значение переменной word, в котором удалена выбранная буква.*

После выхода из цикла переменная *word* становится пустой строкой, а в переменной *ana* сформирована анаграмма.  
Вторая часть программы связана с организацией игры с пользователем:

* *на экран выводится информация, касающаяся самой игры, анаграмма и предложение найти исходное слово;*
* *в цикле ожидается и анализируется ответ пользователя;*
* *если пользователь дал неправильнрый ответ, цикл продолжается – на экран выводится предложение попробовать найти правильное слово еще раз;*
* *если пользователь ответил правильно или решил выйти из игры – цикл заканчивается;*
* *если ответ был правильный – на экран выводится поздравление игроку;*
* *программа благодарит участников за игру и завершает работу.*

 [*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)

#### *2 Диапазоны*

Диапазоны (англ. ranges), как новый тип последовательностей *range*, появились в языке Python, начиная с версий 3.x. До этого функция *xrange()*, которая в 3.x не используется, возвращала последовательность чисел класса *list*.  
Для создания элементов класса *range* необходимо использовать конструктор – встроенную функцию *range([start,] stop [, step])* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#range) лаб. раб. №1), которая возвращает неизменяемую виртуальную последовательность чисел от *start* до *stop - 1* с шагом *step*. Значениями по умолчанию для необязательных аргументов *start*и *step* являются соответственно 0 и 1:  
  
>>> r= range (10)  
>>> r  
range(0, 10)  
>>> for \_ in r:  
print (\_, end= ' ' )  
0 1 2 3 4 5 6 7 8 9  
  
>>> r= range (-5,6)  
>>> r  
range(-5, 6)  
>>> for \_ in r:  
print (\_, end= ' ' )  
-5 -4 -3 -2 -1 0 1 2 3 4 5  
  
>>> r= range (1,12,2)  
>>> r  
range(1,12,2)  
>>> for \_ in r:  
print (\_, end= ' ' )  
1 3 5 7 9 11  
  
Можно представить числа, формируемые функцией *range()*, в виде последовательности определенного типа, использовав соответствующий конструктор, например, в виде списка:  
  
>>> list ( range (1,12,2))  
[1, 3, 5, 7, 9, 11],  
  
или в виде кортежа:  
  
>>> tuple ( range (20,10,-3))  
(20, 17, 14, 11).  
  
Если *stop* не превышает *start*, то формируется пустая последовательность чисел, логическим значением которой является False:  
  
>>> bool ( range (1,1))  
False  
  
Если *step = 0*:  
  
>>> r= range (1,5,0)  
ValueError: range() arg 3 must not be zero,  
  
возникает исключение: 3-й аргумент функции range() не может быть 0.  
Будучи неизменяемой последовательностью:  
  
>>> from collections import Sequence, MutableSequence  
>>> isinstance (range(7), Sequence)  
True,  
>>> isinstance (range(7), MutableSequence)  
False,  
  
тип *range*, как и типы *list* и *tuple*, поддерживает все общие операции, свойственные неизменяемым последовательностям (см. [подраздел 1.4.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2) лаб. раб. №3), кроме операции конкатенации ("+"). Например, можно получить значение элемента последовательности по его индексу:  
  
>>> range (7)[5]  
5.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)

#### *3 Двоичные последовательности*

Для работы с двоичными данными в языке Python имеются встроенные типы данных *bytes* и *bytearray*, которые поддерживаются типом *memoryview*, использующим буферный протокол (buffer protocol) для доступа к другим, находящимся в памяти, двоичным объектам без необходимости их копирования.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)

#### *3.1 Тип bytes*

Объекты *bytes* являются неизменяемыми последовательностями одиночных байтов. Поскольку многие двоичные протоколы базируются на ASCII-кодировании текста, объкты *bytes* предлагают несколько методов, которые допускают работу только с ASCII-совместимыми данными и тесно связаны со строковыми объектами.  
Во-первых, синтаксис литералов *bytes* во многом совпадает с синтаксисом строковых литералов, за исключением того, что добавляется префикс *b*:

* *использование одиночных кавычек:  
    
  >>> b'in"single" quotes'  
  b'in"single" quotes';*
* *использование двойных кавычек:  
    
  >>> b"in 'double' quotes"  
  b"in 'double' quotes";*
* *использование тройных кавычек:  
    
  >>> b'''3 single quotes'''  
  b'3 single quotes',  
  >>> b"""3 double quotes"""  
  b'3 double quotes'.*

Только ASCII-символы разрешается использовать в литералах объекта *bytes* (независимо от объявленного кодирования исходного кода):  
  
>>> b'кириллица'  
SyntaxError: bytes can only contain ASCII literal characters.  
  
Поэтому любые значения литералов *bytes*, превыщающие 127, должны быть преобразованы в литералы *bytes* с использованием соответствующих ESC-последовательнвостей.  
Так же как строковые литералы, литералы *bytes* могут использовать префикс *r* для экранирования ESC-последовательностей. Литералы и строк и объектов *bytes* в общем случае могут иметь следующие варианты префикса: *r* | *R* | *b* | *B* | *br* | *Br* | *bR* | *BR*.  
Хотя литералы и представление объектов *bytes* базируются на ASCII-тексте, они на самом деле ведут себя как незменяемые последовательности целых чисел, значение каждого из которых *x* находится в пределах 0 <= x < 256. При попытках выйти за это органичение вызывет ошибку *ValueError*.  
Помимо использования литеральной формы объекты *bytes* также можно создавать с помощью конструктора класса – встроенной функции *bytes([source[, encoding[, errors]]])* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#bytes) лаб. раб. №1). Необязательный аргумент *source* используется для указания одного из нескольких вариантов создания объекта *bytes*:

* *если аргументом является целое число – создается заполненный нулями (null-байтами) объект bytes указанной длины:  
    
  >>> bytes (8)  
  b'\x00\x00\x00\x00\x00\x00\x00\x00';*
* *если аргументом является итерабельный объект – создается объект bytes из последовательности целых чисел указанного объекта, которые должны находится в диапазоне 0 <= x < 256:  
    
  >>> bytes ( range (15)  
  b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e';*
* *если аргументом является строка – она преобразуется в объект bytes с использованием метода str.encode() (см.*[*подраздел 2.6*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.6)*лаб. раб. №2), при этом обязательно должен быть указан аргумент encoding и опционально errors:  
    
  >>> bytes ( 'ы' , 'utf-8' )  
  b'\xd1\x8b';*
* *если аргументом является объект, применяющий буферный протокол, то для инициализации объекта bytes используются двоичные данные из буфера;*
* *если аргумент не указан – создается объект bytes нулевой длины:  
    
  >>> bytes ()  
  b''.*

Отметим, что для объекта *b* типа *bytes* *b[0]* будет целым числом, в то время как *b[0:1]* объектом *bytes* длины 1:  
  
>>> b= bytes ( range (5))  
>>> b[0]  
0,  
>>> b[0:1]  
b'\x00'.  
  
Этим объекты типа *bytes* отличаются от текстовых строк, где индексное значение и срез строки длиной 1 означают строку длины 1:  
  
>>> s= "cat"  
>>> s[0]  
'c',  
>>> s[0:1]  
'c'.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)

#### *3.2 Тип bytearray*

Объекты *bytearray* являются изменяемым вариантом объектов *bytes*, т.е. представляет собой изменяемую последовательность целых чисел *x* в диапазоне *0 <= x < 256*.  
Литеральные формы создания объектов типа *bytearray* не используются. Эти объекты могут быть созданы с помощью конструктора класса – встроенной функции *bytearray([source[, encoding[, errors]]])* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#bytearray) лаб. раб. №1). Необязательный аргумент *source* используется для указания нескольких вариантов создания объекта *bytearray*:

* *если аргументом является целое число – создается заполненный нулями (null-байтами) объект bytearray указанной длины:  
    
  >>> bytearray (5)  
  bytearray(b'\x00\x00\x00\x00\x00');*
* *если аргументом является итерабельный объект – создается объект bytes из последовательности целых чисел указанного объекта, которые должны находится в диапазоне 0 <= x < 256:  
    
  >>> bytearray ([7,255,0])  
  bytearray(b'\x07\xff\x01');*
* *если аргументом является строка – она преобразуется в объект bytearray с использованием метода str.encode(), при этом обязательно должен быть указан аргумент encoding и опционально errors:  
    
  >>> bytearray ( 'cat' , 'utf-8' )  
  bytearray(b'cat');*
* *если аргументом является объект, применяющий буферный протокол, двоичные данные из буфера используются для инициализации объекта bytearray;*
* *если аргумент не указан – создается объект bytearray нулевой длины:  
    
  >>> bytearray ()  
  bytearray(b'').*

 [*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)

#### *3.3 Операции над объектами типов bytes и bytearray*

Оба типа – и *bytes*, и *bytearray*, являясь последовательностями:  
  
>>> from collections import Sequence  
>>> isinstance (bytes(2), Sequence)  
True,  
>>> isinstance (bytearray(2), Sequence)  
True,  
  
поддерживают все операции, присущие базовым последовательностям (см. [подраздел 1.4.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2) лаб. раб. №3).  
Благодаря общему использованиюю ASCII текста, как базиса для двоичного протокола, объекты типов *bytes*) и *bytearray*) используют почти все методы, присущие текстовым строкам (см. [подраздел 2.6](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.6) лаб. работы №1), за исключением:

* *метода str.encode(), который преобразует текстовые строки в объекты bytes;*
* *методов str.format() и str.format\_map(), которые используются для форматирования текста при выводе на экран;*
* *методов str.isidentifier(), str.isnumeric(), str.isdecimal() и str.isprintable(), которые используются для проверки свойств текстовых строк и обычно не применяются в двоичных протоколах.*

Все остальные строковые методы поддерживаются, хотя иногда с небольшими различиями в функциональности:

* *Методы объектов типов bytes и bytearray не принимают строки, как аргументы своих методов, точно так же, как строки не принимают объекты bytes в качестве своих аргументов:  
    
  >>> a= 'cat'  
  >>> a.replace( 'a' , 'u' )  
  'cut',  
  >>> b= b'cat'  
  >>> b.replace( b'a' , b'u' )  
  b'cut';*
* *в операциях поиска (in, count(), find(), index(), rfind() и rindex()) могут быть использованы как целые числа в диапазоне от 0 до 255 включительно, так и последовательности типов bytes и bytearray.*

Экземпляры как типа *bytes*, так и типа *bytearray* поддерживают метод *decode(encoding="utf-8", errors="strict")*, который является обратным по отношению к методу *str.encode()* и возвращает строку, являющуюся результатом декодирования заданной последовательности байтов. По умолчанию кодировка имеет значение 'utf-8', обработка ошибки – "strict", что означает выработку исключения UnicodeError при обнаружении ошибки кодирования, другими возможными значениями являются: 'ignore', 'replace' и 'xmlcharrefreplace', а также любое другое имя, которое зарегистрировано в codecs.register\_error. Приведем примеры использования методов *encode()* для строк и *decode()* для объектов *bytes*:  
  
>>> 'cat' .encode(encoding= 'utf-8' )  
b'cat',  
>>> b'cat' .decode()  
'cat',  
>>> b= "питон" .encode()  
>>> b  
b'\xd0\xbf\xd0\xb8\xd1\x82\xd0\xbe\xd0\xbd',  
>>> b.decode()  
'питон'.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)

### *Индивидуальные задания*

1 Разработать программу на языке Python, которая выполняет следующее:

* *1.1 Создает кортеж a\_tuple, элементами которого являются объекты следующих типов (см. колонку "Тип" табл. №1):*
  + *1 – целое число;*
  + *2 – число с плавающей точкой;*
  + *3 – логическое значение;*
  + *4 – строка;*
  + *5 – список;*
  + *6 – кортеж.*
* *1.2 Проверяет, является ли объект a\_tuple:*
  + *контейнером;*
  + *итератором;*
  + *итерабельным объектом;*
  + *последовательностью;*
  + *изменяемой последовательностью.*
* *1.3 Изменить значение одного из элементов объекта a\_tuple путем преобразования его сначала в список, а затем снова в кортеж.*
* *1.4 Создает объект a\_range согласно колонке "Диапазон" табл. №1.*
* *1.5 Выводит на экран значения объекта a\_range, используя (см. колонку "Вывод" табл. №1):*
  + *1 – оператор for;*
  + *2 – конструктор списка;*
  + *3 – конструктор кортежа.*

2. Разработать программу, которая:

* *2.1 Выполняет все функции программы "ana.py" (см.*[*подраздел 1.3*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.3)*). При этом формирует до пяти анаграмм средней сложности (степень сложности определяет студент).*
* *2.2 Расширяет функциональность программы следующим образом (см. колонку "Вариант" табл. №1):*
  + *Вариант 1 – если пользователь дал правильное решение, то ему предоставляется возможность выбрать более сложный вариант анаграммы (всего три уровня сложности);*
  + *Вариант 2 – если пользователь дал неправильное решение, то ему предоставляется возможность выбрать более легкий вариант анаграммы (всего три уровня сложности);*
  + *Вариант 3 – пользователю предоставляется возможность получить подсказку: первую букву слова, затем слудующую за ней и т.д. (всего три подсказки);*
  + *Вариант 4 – пользователю предоставляется возможность получить подсказку: последнюю букву слова, затем предыдущую и т.д. (всего три подсказки);*
  + *Вариант 5 – пользователю предоставляется возможность получить следующую подсказку: по его запросу в виде строки, например, \*\*\*а\*\*, означающей – содержит ли исходное слово букву "а" в указанном месте слова, или нет. Программа отвечает утвердительно, если это так, и отрицательно в противном случае (всего три подсказки).*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица 1 – Перечень индивидуальных заданий | | | | |
| **Номер п/п** | **Тип элемента** | **Диапазон** | **Вывод** | **Вариант** |
| 1 | 1,2,3,5 | 1,25 | 1 | 1 |
| 2 | 2,3,5,6 | 2,32,2 | 2 | 2 |
| 3 | 3,4,5,6 | -10,10 | 3 | 3 |
| 4 | 1,6,3,5 | -7,21,3 | 1 | 4 |
| 5 | 2,5,4,3 | 11 | 2 | 5 |
| 6 | 1,2,3,5 | 1,60,5 | 3 | 1 |
| 7 | 2,3,5,6 | 20,0,-2 | 1 | 2 |
| 8 | 3,4,5,6 | 1,25 | 2 | 3 |
| 9 | 1,6,3,5 | 2,32,2 | 3 | 4 |
| 10 | 2,5,4,3 | -10,10 | 1 | 5 |
| 11 | 1,2,3,5 | -7,21,3 | 2 | 1 |
| 12 | 2,3,5,6 | 11 | 3 | 2 |
| 13 | 3,4,5,6 | 1,60,5 | 1 | 3 |
| 14 | 1,6,3,5 | 20,0,-2 | 2 | 4 |
| 15 | 2,5,4,3 | 1,25 | 3 | 5 |
| 16 | 1,2,3,5 | 2,32,2 | 1 | 1 |
| 17 | 2,3,5,6 | -10,10 | 2 | 2 |
| 18 | 3,4,5,6 | -7,21,3 | 3 | 3 |
| 19 | 1,6,3,5 | 11 | 1 | 4 |
| 20 | 2,5,4,3 | 1,60,5 | 2 | 5 |

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#beg)