# ЛАБОРАТОРНАЯ РАБОТА №2

## *Тема: Модули. Строки*

## *Цель: Рассмотрение способов подключения модулей, а также функций и методов работы с текстом*

### *ОГЛАВЛЕНИЕ*

[***1 Использование стандартных модулей***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1)[***1.1 Подключение модулей***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1.1)[***1.2 Модуль builtins***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1.2)[***1.3 Модуль sys***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1.3)[***1.4 Модуль string***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1.4)[***2 Строки***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2)[***2.1 Задание строк в виде литералов***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.1)[***2.2 Задание строк с помощью конструктора str()***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.2)[***2.3 Операции со строками***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.3)[***2.4 Индексация строк***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.4)[***2.5 Срезы строк***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.5)[***2.6 Встроенные функции строк***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.6)[***2.7 Методы строк***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.7)[***Индивидуальные задания***](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#ind)

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *1 Использование стандартных модулей*

Модуль на языке Python – это файл с расширением *.py*, содержащий операторы и определения функций и классов, как правило, общей области применения. Программы на языке Python тоже представляют собой файлы с расширением *.py*, но в отличие от модулей, которые используются для подключения к другим программам, предназначены для непосредственного выполнения.  
При импорте модуля, т.е. подключении модуля к программе, интерпретатор ищет файл с именем, например *my\_module.py*, сначала в текущем каталоге, затем в каталогах, указанных в переменной окружения PYTHONPATH, затем в зависящих от платформы путях, заданных по умолчанию, а также в специальных файлах с расширением *.pth*, которые находятся в стандартных каталогах.  
Программист может внести изменения в PYTHONPATH и в *.pth*, добавив туда свой путь. Каталоги, в которых осуществляется поиск, указаны в переменной *sys.path* (см. [подраздел 1.3](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1.3)).  
Большие программы, как правило, состоят из стартового файла – файла верхнего уровня, и набора файлов-модулей. Главный файл выполняет функции управления программой.  
В то же время модуль – это не только физический файл. Модуль представляет собой набор компонентов. В этом смысле модуль – это пространство имен (англ. namespace), и все имена внутри модуля называются атрибутами – такими, например, как функции и переменные. Для выяснения имен, определенных в модуле, используется встроенная функция *dir()* (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#dir) лаб. раб. №1). Она возвращает отсортированный список строк.  
Python распространяется с библиотекой стандартных модулей. Библиотека включает в себя более 200 модулей, которые выполняют платформенно-зависимую поддержку таких задач, как: интерфейс к операционной системе, управление объектами, поиск, сеть и Internet, GUI и т.д.  
Полный список стандартных модулей можно найти на <http://docs.python.org/3/library/> или на [http://docs.python.org/3/py-modindex.htm](http://docs.python.org/3/py-modindex.html).  
В данном практикуме рассмотрены следующие стандартные модули:

* *builtins (см.*[*подраздел 1.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1.2)*);*
* *sys (см.*[*подраздел 1.3*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1.3)*);*
* *string – содержат описания строковых констант (см.*[*подраздел 1.4*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1.4)*);*
* *collections и collections.abc – содержат описания ABC-классов (см.*[*раздел 1*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1)*лаб. раб. №3), которые не являются встроенными типами и, следовательно, не описаны в модуле builtins;*
* *random (см.*[*подраздел 1.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1.2)*лаб. раб. №4);*
* *cgitb (см.*[*подраздел 3.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB7/lab.htm#3.1)*лаб. раб. №7);*
* *cgi (см. подразделы*[*3.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB7/lab.htm#3.2)*и*[*3.3*](https://py-khpi.github.io/!nfo/doc/lab/LAB7/lab.htm#3.3)*лаб. раб. №7);*
* *os (см. подразделы*[*3.3*](https://py-khpi.github.io/!nfo/doc/lab/LAB7/lab.htm#3.3)*лаб. раб. №7 и*[*1.4*](https://py-khpi.github.io/!nfo/doc/lab/LAB8/lab.htm#1.4)*лаб. раб, №8).*

Помимо встроенных модулей в программах могут быть также использованы модули, разработанные пользователями (см. [раздел 3](https://py-khpi.github.io/!PYTHON/LAB6/lab.htm#3) лаб. раб. №6).

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *1.1 Подключение модулей*

Имеется несколько способов подключения модуля:

* *с помощью оператора import, за которым следует название модуля:  
    
  >>> import some\_module.  
    
  В этом случае обращение к отдельным компонентам модуля, например, вызов функций func1() и func2(), описанных в модуле, выполняется с использованием префикса, представляющего собой имя модуля some\_module:  
    
  >>> some\_module.func1()  
  >>> some\_module.func2();*
* *с помощью оператора from, за которым следует название модуля, оператор import и название компонента модуля (если компонентов несколько, они перечисляются через запятую):  
    
  >>> from some\_module import func1().  
    
  В этом случае вызов функции осуществляется по ее имени без использованя префикса модуля:  
    
  >>> func1().  
    
  При таком варианте подключения модуля существует опасность конфликта пространств имен, если в программе уже используется имя func1. Чтобы этого избежать можно переименовать имя компонента модуля:  
    
  >>> from some\_module import func1 as my\_func1;*
* *с помощью оператора from, за которым следует название модуля, оператор import и \*:  
    
  >>> from any\_module import \*.  
    
  В этом случае к пространству имен программы подключаются все имена модуля.*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *1.2 Модуль builtins*

Модуль *builtins* обеспечивает прямой доступ ко всем встроенным идентификаторам языка Python: встроенным функциям и встроенным константам (см. подразделы [8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#8) и [8.2](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#8.2) лаб. раб. №1). Загружается автоматически при запуске интерпретатора и поэтому обычно не требует явной загрузки.  
Однако в некоторых случаях, например, когда пользователь описывает функцию *open()*, которая вызывает встроенную функцию *open()*, загрузить модуль *builtins* необходимо. При этом вызов встроенной функции осуществляется с использованием имени модуля: *builtins.open()*.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *1.3 Модуль sys*

Модуль *sys* обеспечивает доступ к некоторым переменным и функциям, которые взаимодействуют с интерпретатором языка Python, что позволяет получить после подключения модуля  
  
>>> import sys.  
  
такие данные:

* *sys.builtin\_module\_names – указывает кортеж строк (описание кортежей см. в*[*разделе 1*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1)*лаб. раб. №4), содержащий имена всех доступных модулей:  
    
  >>> import sys  
  >>> sys.builtin\_module\_names  
    
  ('\_ast', '\_bisect', '\_blake2', '\_codecs', '\_codecs\_cn', '\_codecs\_hk', '\_codecs\_iso2022', '\_codecs\_jp', '\_codecs\_kr', '\_codecs\_tw', '\_collections', '\_csv', '\_datetime', '\_functools', '\_heapq', '\_imp', '\_io', '\_json', '\_locale', '\_lsprof', '\_md5', '\_multibytecodec', '\_opcode', '\_operator', '\_pickle', '\_random', '\_sha1', '\_sha256', '\_sha3', '\_sha512', '\_signal', '\_sre', '\_stat', '\_string', '\_struct', '\_symtable', '\_thread', '\_tracemalloc', '\_warnings', '\_weakref', '\_winapi', 'array', 'atexit', 'audioop', 'binascii', 'builtins', 'cmath', 'errno', 'faulthandler', 'gc', 'itertools', 'marshal', 'math', 'mmap', 'msvcrt', 'nt', 'parser', 'sys', 'time', 'winreg', 'xxsubtype', 'zipimport', 'zlib');*
* *sys.exec\_prefix – указывает каталог установки интерпретатора Python:  
    
  >>> sys.exec\_prefix  
  'C:\\Users\\ZYKOV\\AppData\\Local\\Programs\\Python\\Python36-32' ;*
* *sys.executable – указывает путь к интерпретатору Python:  
    
  >>> sys.executable  
  'C:\\Users\\ZYKOV\\AppData\\Local\\Programs\\Python\\Python36-32\\pythonw.exe';*
* *sys.float\_info – указывает информацию о типе данных float:  
    
  >>> sys.float\_info  
  sys.float\_info(max=1.7976931348623157e+308, max\_exp=1024, max\_10\_exp=308, min=2.2250738585072014e-308, min\_exp=-1021, min\_10\_exp=-307, dig=15, mant\_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1) ;*
* *sys.float\_repr\_style – указывает информацию о применении встроенной функции repr() для типа float;*
* *sys.getdefaultencoding() – возвращает используемую кодировку:  
    
  >>> from sys import getdefaultencoding as encoding  
  >>> encoding()  
  'utf-8';*
* *sys.getfilesystemencoding() – возвращает кодировку файловой системы:  
    
  >>> sys.getfilesystemencoding()  
  'mbcs' # multibyte character set ;*
* *sys.int\_info – информация о типе int:  
    
  >>> sys.int\_info  
  sys.int\_info(bits\_per\_digit=15, sizeof\_digit=2) ;*
* *sys.maxsize – максимальное значение числа типа Py\_ssize\_t (2\*\*31 на 32-битных и 2\*\*63 на 64-битных платформах);*
* *sys.maxunicode – максимальное число бит для хранения символа Unicode (Python 3.6.0 – 1114111 (0x10FFFF));*
* *sys.modules – словарь имен загруженных модулей;*
* *sys.path – содержит список строк с именами каталогов, в которых происходит поиск модулей. Он инициализируется из значения переменной окружения PYTHONPATH и встроенного значения по умолчанию:  
    
  >>> sys.path  
    
  ['',  
  'C:\\Users\\ZYKOV\\AppData\\Local\\Programs\\Python\\Python36-32\\Lib\\idlelib', 'C:\\Users\\ZYKOV\\AppData\\Local\\Programs\\Python\\Python36-32\\python36.zip', 'C:\\Users\\ZYKOV\\AppData\\Local\\Programs\\Python\\Python36-32\\DLLs', 'C:\\Users\\ZYKOV\\AppData\\Local\\Programs\\Python\\Python36-32\\lib', 'C:\\Users\\ZYKOV\\AppData\\Local\\Programs\\Python\\Python36-32', 'C:\\Users\\ZYKOV\\AppData\\Local\\Programs\\Python\\Python36-32\\lib\\site-packages'] ;*
* *sys.version – указывает версию языка Python и другие данные (это та же информация, что выводиптся в первой строке интерактивного окна Python Shell (см.*[*подраздел 1.1"*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#1.1)*лаб. раб. №1)):  
    
  >>> sys.version  
    
  '3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)]';*
* *sys.version\_info – указывает кортеж, содержащий пять компонентов номера версии:  
    
  >>> version\_info  
  sys.version\_info(major=3, minor=6, micro=0, releaselevel='final', serial=0).*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *1.4 Модуль string*

Модуль *string* содержит полезные классы и методы, а также следующие строковые константы:

* *string.ascii\_letters – содержит заглавные и строчные буквы латинского алфавита:  
    
  >>> import string # подключение модуля string  
  >>> string.ascii\_letters  
  'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';*
* *string.ascii\_lowercase – содержит строчные буквы латинского алфавита;*
* *string.ascii\_uppercase – содержит заглавные буквы латинского алфавита;*
* *string.digits – содержит цифры:  
    
  >>> string.digits  
  '0123456789';*
* *string.hexdigits – содержит символы для 16-ричных цифр:  
    
  >>> string.hexdigits  
  '0123456789abcdefABCDEF';*
* *string.octdigits – содержит восьмиричные цифры:  
    
  >>> string.octdigits  
  '01234567';*
* *string.punctuation – содержит ASCII-символы, которые считаются символами пунктуации:  
    
  >>> string.punctuation  
  '!"#$%&\'()\*+,-./:;<=>?@ [\\]^\_`{|}~';*
* *string.printable – содержит символы, которые считаются печатаемыми. Они всключают символы, содержащиеся в константах digits, letters, punctuation и whitespace;*
* *string.whitespace – содержит все символы, которые считаются разделительными (whitespace). В большинстве систем это символы пробела (space), \t – табуляции (tab), \n – новой строки (linefeed), \r – возврата каретки (return), \x0b – вертикальной табуляции (vertical tab) и \x0c – перевода на новую страницу при печати (formfeed ):  
    
  >>> string.whitespace  
  ' \t\n\r\x0b\x0c';*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *2 Строки*

Строки (англ. strings) в языке Python – неизменяемые последовательности (см. [подраздел 1.4](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4) лаб. раб. №3) символов в кодировке Unicode, которые имеют тип (класс) *str* и используются для хранения и представления текстовой информации, т.е. любых данных, которые могут быть представлены в текстовой форме.  
Как и для большинства других объектов, для создания объектов класса *str*, т.е. строк, имеется два варианта – либо задав значение в виде строкового литерала (см. [подраздел 2.1](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.1)), либо с помощью конструктора класса – встроенной функции *str()* (см. [подраздел 2.2](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.2)).

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *2.1 Создание строк с помощью литералов*

Строки можно задавать в виде литералов с использованием:

* *одиночных кавычек:  
    
  >>> print ( 'Строка может включать "двойные" кавычки' )  
  Строка может включать "двойные" кавычки;*
* *двойных кавычек:  
    
  >>> print ( "Эта строка может включать 'одиночные' кавычки" )  
  Эта строка может включать 'одиночные' кавычки ;*
* *тройных кавычек, которые могут быть составлены:*
  + *из трех одиночных кавычек:  
      
    >>> print ( '''Используются 'одиночные' кавычки''' )  
    Используются 'одиночные' кавычки ;*
  + *из трех двойных кавычек:  
      
    >>> print ( """Используются "двойные" кавычки""" )  
    Используются "двойные" кавычки.*
* *префикса f и кавычек любого вида. При этом литерал содержит поля замещения, указанные в фигурных скобках, которые представляют выражения, значения которых определяются при выполнении программы и подставляются в литерал:  
    
  >>> name= 'Том'  
  >>> f 'Его звать {name}'  
  'Его звать Том'*

Для совместимости с версиями 2.x языка Python, начиная с версии 3.3, разрешено использование префикса "u", который для версий 3.3 и выше ни на что не влияет:  
  
>>> u'cat'  
'cat'  
  
Для помещения в строку символов, выполняющих в языке Python служебные функции (**'**, **"** и **\**, а также символов, связанных с работой клавиатуры, используются наборы символов, называемые ESC-последовательностями. В таблице 1 приведено соотношение между ESC-последовательностями и символами, вставляемыми в строку при выводе вместо них.

|  |  |
| --- | --- |
| Таблица 1 – Соотношение между ESC-последовательностями и символами | |
| **ESC-последовательность** | **Символ** |
| \' | ' |
| \" | " |
| \\ | \ |
| \b | забой |
| \n | новая строка |
| \f (\x0c) | новая страница |
| \r | возврат каретки |
| \t | горизонтальная табуляция |
| \v (\x0b) | вертикальная табуляция |
| \a | звонок |
| \uhhhh | 16-битовый символ Юникода в 16-ричном представлении |
| \Uhhhh… | 32-битовый символ Юникода в 16-ричном представлении |
| \xhh | 16-ричное значение символа |
| \ooo | 8-ричное значение символа |
| \0 | символ Null |

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)  
Отметим, что для воспроизведения звука по *\a* необходимо запустить программу не из среды разработки, а по двойному щелчку по файлу.  
Ниже приводится пример использования ESC-последовательностей для задания горизонтальной табуляции и новой строки:  
  
>>> print ( "It's cat\tand that is dog\nnew string" )  
It's cat         and that is dog  
new string  
  
Для экранирования большинства ESC-последовательностей необходимо перед строкой указать букву *r* или *R*. При этом создаются исходные строки ("raw strings"):  
  
>>> print ( "1\t2\n\x33" )  
1        2  
3  
>>> print ( r"1\t2\n\x33" )  
1\t2\n\x33  
  
Длинные строки можно располагать на нескольких рядках кода, используя символ обратного слэша ("\"):  
  
>>> s= "word1 " \  
"word2 " \  
"word3"  
>>> s  
word1 word2 word3.  
Одной из особенностей тройных кавычек является их применение для задания многострочного текста с соблюдением всех разделительных символов, при этом ESC-последовательности экранируются:  
  
>>> print ( '''  
"Из всех вещей, что нас окружают,  
ничто есть величайшее"  
Леонардо да Винчи ''' )  
  
"Из всех вещей, что нас окружают,  
ничто есть величайшее"  
Леонардо да Винчи  
  
Другой особенностью тройных кавычек являетеся их использование для документирования функций (см. [подраздел 1.1](https://py-khpi.github.io/!nfo/doc/lab/LAB6/lab.htm#1.1)лаб. раб. №6) и классов (см. [подраздел 1.1](https://py-khpi.github.io/!nfo/doc/lab/LAB11/lab.htm#1.1) лаб. раб. №11).  
Модуль string (см. [подраздел 1.4](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#1.4)) содержит строковые константы: все заглавные (uppercase) буквы, строчные (lowercase) буквы , цифры, символы пунктуации (punctuation) и разделительные символы (whitespace).  
В табл. 2 представлены все разделительные символы.

|  |  |  |
| --- | --- | --- |
| Таблица 2 – Разделительные символы | | |
| **ESC-последовательность** | **ASCII-наименование** | **Наименование** |
|  | SP | пробел |
| '\n' | NL | новая строка |
| '\r ' | CR | возврат каретки |
| '\t ' | HT | горизонтальная табуляция |
| '\v' (\x0b) | VT | вертикальная табуляция |
| '\f' (\x0c) | FF | новая форма |

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *2.2 Создание строк с помощью конструктора класса*

Строки могут быть созданы также с помощью конструктора класса – встроенной функции *str(object='', encoding='utf-8', errors='strict')*, которая возвращает представление объекта *object* в виде строки:

* *если аргумент функции object опущен, то возвращается пустая строка:  
    
  >>> str ()  
  '';*
* *если аргументы encoding и errors не заданы, то функция str(object) с помощью метода \_\_str\_\_() возвращает представление объекта в печатном виде:*
  + *если аргументом функции является значение типа str, то возвращается строка с этим значением:  
      
    >>> str ( 'cat' )  
    'cat';*
  + *если аргументом функции является значение другого типа, то оно преобразуется в тип str и возвращается в виде строки:  
      
    >>> str (1.25-e1)  
    '0.125';*
* *если для объекта object не определен метод \_\_str\_\_(), то неявно вызывается встроенная функция repr() (см.*[*подраздел 2.5*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.5)*)  
    
  >>> str ( type (1.33))  
  "<class 'float'>";*
* *если хотя бы один из аргументов encoding или errors задан, аргумент object должен быть двоичным объектом (т.е. иметь тип bytes или bytearray, (см.*[*раздел 3*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3)*лаб. раб. №4). В этом случае функция str(bytes, encoding, errors) эквивалентна методу decode(encoding, errors) объекта bytes (см.*[*подраздел 3.3*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.3)*лаб. раб. №3) и осуществляет декодирование двоичной последовательности в строку с использованием кодека, заданного арнументом encoding (значение по умолчанию, равное 'utf-8', берется из sys.getdefaultencoding()) и обработчика ошибок (аргумент errors). Если encoding неизвестен, вырабатывается исключение LookupError. Аргумент errors задает режим обработки ошибок: если его значение равно 'strict' (значение по умолчанию), то при ошибке возникает исключение ValueError, при значении 'ignore' ошибки игнорируются и при значении 'replace' – вместо символа, который не мог быть декодирован вставлятся символ U\FFFD.*

Число байтов, необходимых для хранения Unicode строк в памяти (для Python 3.4) зависит от наибольшего значения кода символа в строке:

* *для символов, которые могут быть выражены в ASCII и Latin1 кодировке (U+0000-U+00FF), используется 1 байт на код;*
* *для символов, которые могут быть быражены в BMP кодировке (U+0000-U+FFFF), используется 2 байта на код;*
* *для символов, которые не могут быть быражены в BMP кодировке (U+10000-U+10FFFF), используется 4 байта на код.*

Укажем, что BMP (Basic Multilangual Plane) означает базовую многоязыковую плоскость Юникода (первые 65536 символов), в которую входят и символы кириллицы.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *2.3 Операции со строками*

Над строками можно выполнять следующие операции:

* *x in S – равно True, если строка S содержит символ x, иначе равно False (операция, общая для последовательностей, см.*[*подраздел 1.4.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)*лаб. раб. №3) . Может быть использована для проверки вхождения подстроки в строку:  
    
  >>> 'Comp' in 'Computer'  
  True  
  >>> 'comp' in 'Computer'  
  False;*
* *x not in S – равно False, если строка S содержит символ x, иначе равно True (операция, общая для последовательностей, см.*[*подраздел 1.4.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)*лаб. раб. №3):  
    
  >>> 'comp' not in 'Computer'  
  True;*
* *S + T – выполняется операция конкатенации между строками S и T (операция, общая для последовательностей, см.*[*подраздел 1.4.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)*лаб. раб. №3):  
    
  >>> s= 'Операция'  
  >>> t= ' конкатенации'  
  >>> s+t  
  'Операция конкатенации'.  
    
  Python поддерживает для операции конкатенации расширенную форму оператора присваивания (см.*[*подраздел 6.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#6)*лаб. раб. №1):  
    
  >>> s= 'cat'  
  >>> s+= 's'  
  >>> s  
  'cats'.  
    
  Операция конкатенации может быть выполнена без использования операторов "+":  
    
  >>> "cat" " and " "dog"  
  'cat and dog',  
    
  но только в том случае, когда операнды задаються в виде литералов (без использования переменных). Иначе вызывается ошибка:  
    
  >>> s " and " "dog"  
  SyntaxError: invalid syntax*
* *n\*S (S\*n) – выполняется n-кратное копирование строки S (операция, общая для последовательностей, см.*[*подраздел 1.4.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)*лаб. раб. №3):  
    
  >>> s= '0'  
  >>> '1' +6\*s  
  '1000000'.*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *2.4 Индексация строк*

Хотя строки представляют собой неизменяемые последовательности символов, доступ к их отдельным символам допускается. Каждый символ строки имеет свой номер (индекс). Python поддерживает две системы нумерации символов:

* *нумерация слева направо – самый левый символ строки имеет индекс 0, последний – индекс, на 1 меньший числа символов строки;*
* *нумерация справа налево – самый правый (последний) символ строки имеет индекс -1, самый левый – индекс, равный числу символов строки со знаком минус.*

Строки поддерживают операцию получения значения элемента строки по его индексу *s[i]*, где *s* – строка, *i* – номер элемента (операция, общая для последовательностей, см. [подраздел 1.4.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2) лаб. раб. №3):  
  
>>> s= 'Computer'  
>>> s[0]  
'C'  
>>> s[7]  
'r'  
>>> s[-2]  
'e'  
>>> s[8]  
IndexError: string index out of range  
  
При обращении к индексу строки, который превышает номер ее последнего символа, произошла ошибка и интерпретатор сообшает:  
  
IndexError: индекс строки вне диапазона.

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *2.5 Срезы строк*

Помимо обращения к отдельным символам строки (см. [подраздел 2.4](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#2.4)) Python допускает обращение к отдельным последовательностям символов строки, которые называются срезами (англ. slices) строки. В общем случае операции со срезами, которые являются операциями, общими для последовательностей (см. [подраздел 1.4.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2) лаб. раб. №3), представляется следующей конструкцией:  
  
s[i:j:k],  
  
где:

* *s – строка, которая может быть задана переменной или литералом строкового типа;*
* *i – номер символа строки, с которого начинается срез;*
* *j – номер символа строки, который следует после последнего символа среза, т.е. этот символ не входит в срез;*
* *k – интервал перебора номеров строки, например, если k=2, то символы перебираются через один.*

Причем каждый из параметров *i*, *j* или *k* может быть опущен (даже все):

* *если опущен i – нумерация среза начинается с индекса 0;*
* *если опущен j – последним символом среза будет последний символ строки;*
* *если опущен k – подразумевается, что k=1, т.е. символы среза указываются подряд слева направо (для указания последовательности символов справа налево необходимо для k указывать отрицательные значения).*

Срезы с неправильными границами обрабатываются следующим образом:

* *если верхняя граница среза больше длины строки, то она уменьшается до длины строки;*
* *если нижняя граница больше верхней, то возвращается пустая строка*

Примеры срезов:  
  
>>> s2= 'Микропроцессор'  
>>> s2[5:12]  
'процесс'  
>>> s2[:5]  
'Микро'  
>>> '0123456789' [::2]  
'02468'.  
  
В срезах можно использовать нумерацию справа налево:  
  
>>> s2[-9:]  
'процессор'

и выбирать символы среза справа налево:  
  
>>> '0123456789' [-1:-11:-1]  
'9876543210'  
  
Пример задания пустого среза:  
  
>>> '0123456789' [7:2]  
''.  
  
Отметим, что операции со срезом являются общими для последовательностей (см. [подраздел 1.4.2](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)лаб. раб. №3).

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *2.6 Встроенные функции строк*

При работе со строками могут быть использованы следующие встроенные функции (см. [подраздел 8.1](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#8) лаб. раб. №1):

* [*сhr(сode)*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#chr)*– возвращает строку из одного символа, который соответствует числовому значению аргумента, заданному в ASCII или Unicode коде в диапазоне от 0 до 1114111 (0x10FFFF):  
    
  >>> chr (1092)  
  'ф'.  
    
  При превышении диапазона вырабатывается исключение ValueError. Функция сhr(сode) является обратной функции ord(). Она эквивалентна функции uniсhr(сode), которая используется в версиях 2.x языка Python;*
* [*len()*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#len)*– возвращает длину строки (число ее символов):  
    
  >>> len ( 'Python' )  
  6.  
    
  Функция len(s) эквивалентна методу s.\_\_len\_\_():  
    
  >>> 'Python' .\_\_len\_\_()  
  6.  
    
  Функция len() является операцией, общей для последовательностей (см.*[*подраздел 1.4.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)*лаб. раб. №3);*
* [*max()*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#max)*– возвращает символ строки с максимальным числовым значением согласно таблице ASCII или Unicode:  
    
  >>> max ( 'Python' )  
  'y'.  
    
  Функция max() является операцией, общей для последовательностей (см.*[*подраздел 1.4.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)*лаб. раб. №3);*
* [*min()*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#min)*– возвращает символ строки с минимальным числовым значением согласно таблице ASCII или Unicode:  
    
  >>> min ( 'Python' )  
  'P'.  
    
  Функция min() является операцией, общей для последовательностей (см.*[*подраздел 1.4.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)*лаб. раб. №3);*
* [*ord(char)*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#ord)*– возвращает числовое значение строки из одного символа согласно таблице ASCII или символа Unicode:  
    
  >>> ord ( 'ф' )  
  1092.  
  >>> ord ( 'Python' [0])  
  80;*
* [*repr(object)*](https://py-khpi.github.io/!nfo/doc/lab/LAB1/lab.htm#repr)*– возвращает строку, в большинстве случаев содержащую представление объекта, предназначенное для работы интерпретатора. Например, в форме, пригодной для использования в функции eval():  
    
  >>> repr ( 'cat' )  
  "'cat'"  
  >>> eval ( "'cat'" )  
  'cat',  
    
  а значение, возвращаемое функцией str() нельзя использовать как аргумент функции eval():  
    
  >>> str ( 'cat' )  
  'cat'  
  >>> eval ( 'cat' )  
  NameError: name 'cat' is not defined  
    
  В других случаях возвращается строка, содержащая заключенные в угловые скобки данные: наименование типа объекта вместе с дополнительной информацией, часто включающей наименование объекта и его адрес в памяти:  
    
  >>> repr ( type (1.33))  
  "<class 'float'>".*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *2.7 Методы строк*

При работе со строками могут быть использованы следующие методы (*S* – любая строка, которая может быть представлена либо в виде строкового литерала, либо в виде переменной типа *str*):

* *S.capitalize() – возвращает строку s с первой заглавной буквой (если первый символ строки – буква):  
    
  >>> import string.  
  >>> 'cat' .capitalize()  
  'Cat';*
* *S.casefold(...) – возвращает вариант строки S, подходящий для сравнения без учета регистра:  
    
  >>> 'cat' .casefold()  
  'cat',  
    
  >>> 'Cat' .casefold()  
  'cat';*
* *S.center(width [, fillchar]) – возвращает S, выравненную по центру строки шириной width, свободные места слева и справа от s заполняются символами fillchar (по умолчанию пробел), если число заполнителей нечетно, то справа помещается на один заполнитель больше. Если длина строки равна или больше width – возвращает строку S:  
    
  >>> s= 'Python'  
  >>> s.center(12, '\_' )  
  '\_\_\_Python\_\_\_';*
* *S.count( sub[, start[, end]]) – возвращает число встретившихся подстрок sub в строке S[start:end], необязательные аргументы start и end интерпретируются как параметры среза, в примере подсчитывается число нулей в строке, начиная с индекса 8 до конца строки:  
    
  >>> '0000000100100011' .count( '0' ,8)  
  5.  
    
  <="" span="" style="margin-left: 1.5cm;">Метод является операцией, общей для последовательностей (см.*[*подраздел 1.4.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)*лаб. раб. №3);*
* *S.encode([encoding[, errors]]) – возвращает последовательность байтов (см.*[*подраздел 3.1*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.1)*лаб. раб. №4), являющуюся результатом кодирования строки S с помощью кодека, указанного аргументом encoding (по умолчанию – 'utf-8') (см.*[*подраздел 3.3*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#3.3)*лаб. раб. №4). Аргумент errors может быть указан для задания режим обработки ошибки: по умолчанию – 'strict', указывающий, что при возникновении ошибки будет выработано исключение UnicodeEncodeError, другими возможными значениями являются: 'ignore', 'replace' и 'xmlcharrefreplace', а также любое другое имя, которое зарегистрировано в codecs.register\_error;*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

* *S.endswith(T[, start[, end]]) – проверяет, заканчивается ли строка S строкой T, если указаны start[, end], то вместо S берется ее срез:  
    
  >>> 'cats' .endswith( 's' )  
  True  
  >>> 'cat' .endswith( 's' )  
  False;*
* *S.expandtabs([tabsize]) – возвращает строку S, в которой все знаки горизонтальной табуляции заменены пробелами, их число указанно аргументом tabsize (по умолчанию 8):  
    
  >>> 1\t2\t3\t4' .expandtabs(5)  
  '1      2     3      4';*
* *S.find(T[,start[,end]]) – если строка T не найдена в S, возвращает -1, иначе возращает индекс первой позиции в S, с которой совпала строка T. Если указаны start[, end], то вместо S берется ее срез:  
    
  >>> s= 'cat and dog'  
  >>> s.find( 'dog' )  
  8  
  >>> s.find( 'dog' ,0,5)  
  -1;*
* *S.format(\*args, \*\*kwargs) – осуществляет форматирование строк, используя подстановки из args и kwargs;*
* *S.index(T[,start[,end]]) – работает так же, как S.find(T[,start[,end]]), но если строка T не найдена, вырабатывает ислючение ValueError: substring not found (подстрока не найдена):  
    
  >>> s.index( 'pig' )  
  ValueError: substring not found  
    
  Метод является операцией, общей для последовательностий (см.*[*подраздел 1.4.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.4.2)*лаб. раб. №3);*
* *S.isalnum() – возвращает True, если S.isalnum() – возвращает True, если содержит хотя бы один символ и все ее символы содержит хотя бы один символ и все ее символы являются буквами или цифрами, иначе возвращает False:  
    
  >>> '5\_cats' .isalnum()  
  False  
  >>> '5cats' .isalnum()  
  True  
  >>> '' .isalnum()  
  False;*
* *S.isalpha() – возвращает True, если содержит хотя бы один символ и все ее символы являются буквами, иначе возвращает False:  
    
  >>> 'one cat' .isalpha()  
  False  
  >>> 'Cat' .isalpha()  
  True;*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

* *S.isdecimal() – возвращает True, если все символы S являются десятичными символами (decimals), иначе возвращает False. К десятичным символам относятся цифры, а также все символы, которые могут быть использованы для формирования чисел с десятичным основанием, например, с помощью Юникода:  
    
  >>> '23' .isdecimal()  
  True  
  >>> c= '\u0032\u0033'  
  >>> c  
  '23'  
  >>> c.isdecimal()  
  True  
  >>> '' .isdecimal()  
  False;*
* *S.isdigit() – возвращает True, если s содержит хотя бы один символ и все ее символы являются цифрами (digits), иначе возвращает False. Цифры включают десятичные символы и цифры, которые требуют обработки, например, показатели степени:  
    
  >>> c= 'u0032\u00B3\'  
  >>> c  
  '23'  
  >>> c.isdecimal()  
  False  
  >>> c.isdigit()  
  True;*
* *S.isidentifier() – возвращает True, если S является идентификатором, заданным в соответствии с правилами языка:  
    
  >>> 'name\_1' .isidentifier()  
  True  
  >>> '\_' .isidentifier()  
  True  
  >>> '-' .isidentifier()  
  False;*
* *S.islower() – возвращает True, если S содержит хотя бы один символ и все ее символы являются строчными буквами, иначе возвращает False:  
    
  >>> '123' .islower()  
  False  
  >>> 'Cat' .islower()  
  False  
  >>> 'cat' .islower()  
  True;*
* *S.isnumeric() – возвращает True, если все символы строки S являются числовыми символами (numeric characters), иначе возвращает False. Числовые символы включают цифровые символы и все символы, которые в Юникоде формируют числовые значения, например, значение дроби, равное одной пятой:  
    
  >>> c= '\u2155'  
  >>> c  
  '1/5''  
  >>> c.isdecimal()  
  False  
  >>> c.isdigit()  
  False  
  >>> c.isnumeric()  
  True;*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

* *S.isprintable() – возвращает True, если все символы строки S являются печатными или строка пуста, иначе возвращает False. Непечатными являются те символы, которые определены в базе данных Юникода как "Other” (”Другие”) или “Separator” (”Разделители”), исключая символ пробела (ASCII код равен 0x20), который считается печатным:  
    
  >>> c= 'www'  
  >>> c.isprintable()  
  True  
  >>> c= '\u0007'  
  >>> c.isprintable()  
  False  
  >>> '' .isprintable()  
  True;*
* *S.isspace() – проверяет, не является ли S пустой строкой и все ли ее символы являются пробелами, символами табуляции, возврата каретки или новой строки:  
    
  >>> '\t\n\r' .isspace()  
  True  
  >>> ' \t\n\r cat' .isspace()  
  False;*
* *S.istitle() – проверяет, не является ли S пустой строкой и находится ли она в титульном режиме (“title case”) – в этом режиме заглавные буквы могут появится только в начале строки или после некоторых символов, которые не являются буквами. Строчные буквы могут появится только после заглавных:  
    
  >>> 'A cat' .istitle()  
  False  
  >>> 'A Cat' .istitle()  
  True;*
* *S.isupper() – проверяет, не является ли S пустой строкой и все ли ее буквы являются заглавными (небуквенные символы не проверяются):  
    
  >>> 'A Cat' .isupper()  
  False  
  >>> 'A CAT' .isupper()  
  True;*
* *S.join(iterable) – аргумент iterable должен быть итерабельным объектом (см.*[*подраздел 1.2*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#1.2)*лаб. раб. №3), который содержит последовательность строк. Возвращаемое значение представляет собой строку, состоящую из строк объекта iterable, соединенных между собой строкой S. Например, если объект list\_a является списком строк (см.*[*подраздел 2.4*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2.4)*лаб. раб. №3):  
    
  >>> list\_a=[ 'cat' , 'dog' , 'rabbit' ],  
    
  тогда метод join():  
    
  >>> ' and ' .join(list\_a)  
  возвратит строку:  
    
  'cat and dog and rabbit';*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

* *S.ljust(width [, fillchar]) – возвращает S, выравненную по левому краю строки шириной width, свободные места справа от S заполняются символами fillchar (по умолчанию пробел), если длина строки равна или больше width – возвращает строку S:  
    
  >>> s= 'Python'  
  >>> s.ljust(10)  
  'Python    ';*
* *S.lower() – возвращает копию S, в которой все заглавные буквы заменены строчными:  
    
  >>> "It's a Cat" .lower()  
  "it's a cat";*
* *S.lstrip([c]) – возвращает S с удаленными из строки всеми начальными символами, которые указаны в строке c (по умолчанию берется строка, содержащая все разделительные символы (см.*[*табл. 2*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#tabl2)*):  
    
  >>> ' \n \r \tcat \n\t and \t dog' .lstrip()  
  'cat \n\t and \t dog'  
    
  >>> '\_\_\_\_\_cat\_\_\_\_\_\_' .lstrip( '\_' )  
  'cat\_\_\_\_\_\_';*
* *S.partition(sep) – выполняет поиск в строке S первого появления разграничителя строки sep. Если S содержит разграничитель, то возвращается кортеж (pre, sep, post), где pre – часть S до разграничителя, sep – сам разграничитель и post – часть S после разграничителя (кортежи рассмотрены в*[*разделе 1*](https://py-khpi.github.io/!nfo/doc/lab/LAB4/lab.htm#1)*лаб. раб. №4). Если разграничитель не найден – возвращается трехэлементный кортеж (S, '', '') :  
    
  >>> s= 'Cat and dog'  
  >>> s.partition( ' ' )  
  ('Cat', ' ', 'and dog')  
  >>> s.partition( ',' )  
  ('Cat and dog', '', '');*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

* *S.replace(old,new[,max]) – возвращает строку S, в которой осуществлена замена подстрок old подстроками new. По умолчанию делается замена всех найденных подстрок, необязательный аргумент max показывает, сколько подстрок нужно заменить:  
    
  >>> 'Cat, white dog and black dog play together' .replace( 'dog' , 'small dog' ,1)  
  Cat, white small dog and black dog play together;*
* *S.rfind(T[,start[,end]]) – аналогичен методу S.find(T[,start[,end]]), но возвращает начальный индекс последнего вхождения T:  
    
  >>> s= 'cat and small cat'  
  >>> s.find( 'cat' )  
  0  
  >>> s= 'cat and small cat'  
  >>> s.rfind( 'cat' )  
  14;*
* *S.rindex(T[, start[,end]]) – аналогичен методу S.index(T[, start[, end]]), но возвращает индекс последнего совпадения строки T в строке S:  
    
  >>> s= 'cat and dog'  
  >>> s.index( ' ' )  
  3  
  >>> s.rindex( ' ' )  
  7;*
* *S.rjust(width[, fillchar]) – возвращает S, выравненную по правому краю строки шириной width, свободные места слева от S заполняются символами fillchar (по умолчанию пробел), если длина строки равна или больше width – возвращает строку S:  
    
  >>> s= 'Python'  
  >>> s.rjust(10)  
  '    Python';*
* *S.rpartition(sep) – аналогичен методу S.partition(sep), но ищет последенее появление разграничителя:  
    
  >>> 'Cat and dog' .partition( ' ' )  
  ('Cat', ' ', 'and dog')  
  >>> 'Cat and dog' .rpartition( ' ' )  
  ('Cat and', ' ', 'dog');*
* *S.rsplit(sep[,max]) – аналогичен методу S.split(sep[,max]), за исключением того, что если число частей строки S превышает max, то неразделенная часть будет помещена не в конец списка, а в его начало:  
    
  >>> 'Cat and dog' .rsplit( None ,1)  
  ['cat and', 'dog'];*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

* *S.rstrip([C]) – возвращает S с удалением из нее всех конечных символов, которые указаны в строке C (по умолчанию берется строка, содержащая все разделительные символы (см.*[*табл. 2*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#tabl2)*):  
    
  >>> 'cat and dog\n \r' .rstrip()  
  'cat and dog'  
    
  >>> '\_\_\_\_\_cat\_\_\_\_\_\_' .lstrip( '\_' )  
  '\_\_\_\_\_cat';*
* *Ы.split([ыуз[,max]]) – возвращает список строк, полученных путем разделения Ы на части с помощью разделителя sep (см.*[*подраздел 2.4*](https://py-khpi.github.io/!nfo/doc/lab/LAB3/lab.htm#2.4)*лаб. раб. №3). По умолчанию в качестве разделителя используется любой из разделительных символов (см.*[*табл. 2*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#tabl2)*). Необязательный аргумент max определяет максимальное число частей, на которые может делиться строка S – число элементов возвращаемого списка не должно превышать max+1:  
    
  >>> 'Cat, small dog\n and\tbig dog' .split()  
  ['Cat,', 'small', 'dog', 'and', 'big', 'dog']  
  >>> 'Cat\_and\_dog\_' .split( '\_' )  
  ['Cat', 'and', 'dog', '']  
  >>> 'Cat and dog' .split( None ,1)  
  ['Cat', 'and dog'];*
* *S.startswith(T[, start[, end]]) – проверяет, начинаетсяли строка S строкой T, если указаны start[, end], то вместо S берется ее срез:  
    
  >>> 'A cat' .startswith( 'A ' )  
  True  
  >>> 'cat' .startswith( 'A ' )  
  False;*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

* *S.strip([C]) – возвращает s с удаленными из строки всеми начальными и конечными символами, которые указаны в строке C (по умолчанию берется строка, содержащая все разделительные символы (см.*[*табл. 2*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#tabl2)*):  
    
  >>> '\r\n\tcat and dog\r\n' .strip()  
  'cat and dog'  
    
  >>> '\_\_\_\_\_cat\_\_\_\_\_\_' .strip( '\_' )  
  'cat';*
* *S.swapcase() – возвращает копию S, у которой каждый строчный символ заменен заглавным, а каждый заглавный – строчным:  
    
  >>> 'Cat and Dog' .swapcase()  
  'cAT AND dOG';*
* *S.title() – возвращает копию S, у которой первый символ каждого слова является заглавной буквой, а остальные – строчными:  
    
  >>> 'CAT and DOG' .title()  
  'Cat And Dog';*
* *S.translate(new[,drop]) – используется для трансляции или удаления любого символа строки S. Аргумент new должен быть строкой из 256 символов, и каждый символ результата x определяется как new[ord(x)];*
* *S.upper() – возвращает копию s, у которой каждый строчный символ заменен заглавным:  
    
  >>> 'Cat and Dog' .upper()  
  'CAT AND DOG';*
* *S.zfill(w) – возвращает копию S, заполненную слева символами '0' до ширины w:  
    
  >>> '25' .zfill(10)  
  '0000000025'.*

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)

#### *Индивидуальные задания*

Разработать программу на языке Python, которая выполняет следующее:

* *Вводит фамилию, имя и отчество студента в виде одной строки.*
* *Определяет срез этой строки согласно колонке "Срез" табл. 3.*
* *Применяет к строке метод согласно колонке "Метод" табл. 3.*
* *Выводит на экран 5 вопросов, по одному на каждую тему согласно колонке "Темы" табл. 3.  
  Перечень тем:*
  + *1. Операции с числами;*
  + *2. Функции строк;*
  + *3. ABC-классы;*
  + *4. Целые числа;*
  + *5. Задание строк;*
  + *6. Срезы строк;*
  + *7. Методы строк;*
  + *8. Операторы ввода/вывода;*
  + *9. Условный оператор;*
  + *10. Операторы цикла.*
* *Получает ответы и проверяет их правильность.*
* *Подсчитывает сумму баллов и выводит на экран фамилию студента и полученную им оценку.*

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица 3 – Перечень индивидуальных заданий | | | |
| **Номер п/п** | **Срез** | **Метод** | **Темы** |
| 1 | 2:7 | find() | 1,2,3,4,5 |
| 2 | :10 | replace() | 6,7,3,4,5 |
| 3 | 7: | upper() | 6,2,8,4,5 |
| 4 | 2:12:2 | swapcase() | 6,2,3,5,9 |
| 5 | 3:8 | lower() | 2,3,4,6,10 |
| 6 | :6 | center() | 2,3,5,7,8 |
| 7 | 8: | count() | 2,3,4,7,9 |
| 8 | :12,3 | endswith() | 2,3,4,8,10 |
| 9 | 1:5 | index() | 1,2,5,6,7 |
| 10 | :14 | ljust() | 1,2,3,6,10 |
| 11 | 4: | rfind() | 1,2,3,4,9 |
| 12 | 3::2 | rindex() | 6,7,3,4,5 |
| 13 | -8:-1 | rjust() | 6,2,8,4,9 |
| 14 | :-2 | startswith() | 6,8,3,5,9 |
| 15 | 5:-3 | title() | 2,5,4,6,10 |
| 16 | -2-6:-2 | zfill() | 2,4,5,7,8 |
| 17 | -3:1:-1 | replace() | 5,3,4,7,9 |
| 18 | 1:-1 | find() | 2,3,4,8,9 |
| 19 | :-8 | isidentifier() | 4,2,5,6,7 |
| 20 | -4::-2 | islower() | 8,2,3,6,10 |

[*Оглавление*](https://py-khpi.github.io/!nfo/doc/lab/LAB2/lab.htm#beg)