Лекция 1 "Введение в программирование на Python"

часть 3 Строки

Финансовый университет при Правительстве РФ, лектор С.В. Макрушин

v 0.6 08.09.2020

In [1]:

```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v1.css")
HTML(html.read().decode('utf-8'))
```

Out[1]:

Разделы:

-

- к оглавлению

Строки: введение

- к оглавлению

Строки в Python являются неизменяемым типом данных, содержащим массив символов.

In [17]:

```
s = "Hello world" # объявление строки с помощью двойных кавычек (не единственный способ!)
type(s)
```

Out[17]:

```
str
```

In [18]:

```
print(s)
```

```
Hello world
```

In [20]:

```
s2 = 'Hello world' # объявление строки с помощью одинарных кавычек
print(s2)
print('Проверка равенства: ', s == s2)
print('Проверка идентичности: ', s is s2)
```

```
Hello world
Проверка равенства:  True
Проверка идентичности:  False
```

In [21]:

```
a1 = 'ab'
a2 = 'ab'
# тут, как и для малых чисел, работает кэширование:
print('Проверка идентичности коротких строк: ', a1 is a2)
```

```
Проверка идентичности коротких строк:  True
```

In [23]:

```
# длина строки (количество символов)
len(s)
```

Out[23]:

```
11
```

In [10]:

```
# функция str() возвращает строковое представление любого объекта
str(11)
```

Out[10]:

```
'11'
```

In [24]:

```
type(str(11))
```

Out[24]:

```
str
```

In [25]:

```
str([1, 3, 5])
```

Out[25]:

```
'[1, 3, 5]'
```

Специальные символы

- к оглавлению

Использование кавычек в тексте строки.

In [26]:

```
st1 = 'With " double quotes'
print(st1)
```

```
With " double quotes
```

In [27]:

```
st2 = "With ' unary quotes"
print(st2)
```

```
With ' unary quotes
```

- \n - перевод строки;
- \r - возврат каретки;
- \t - знак табуляции;
- \v - вертикальная табуляция;
- \a - звонок;
- \b - забой;
- \f - перевод формата;
- \O - нулевой символ (не является концом строки);
- " - кавычка;
- ' - апостроф;
- \N - восьмеричное значение N. Например, \74 соответствует символу <;
- \xN - шестнадцатеричное значение N. Например, \x6a соответствует символу j;
- \\ - обратный слэш;
- \uxxxx - 16-битный символ Unicode. Например, \u043a соответствует русской букве к;
- \Uxxxxxxxx - 32-битный символ Unicode.

Если после слэша не стоит символ, который вместе со слэшем интерпретируется как спец символ, то слэш сохраняется в составе строки.

In [29]:

```
print('Строка1 \nСтрока2 обра\тный слэш: \\ символ с кодом 6A: \x6A')
```

```
Строка1
Строка2 обра    ный слэш: \ символ с кодом 6A: j
```

In [30]:

```
# обратный слэш может экранировать перевод каретки:
sl = 'строка, введенная \
на нескольких строках'
sl
```

Out[30]:

```
'строка, введенная на нескольких строках'
```

Строку, введенную между утроенными апострофами или утроенными кавычками, можно разместить на нескольких строках, а также одновременно использовать кавычки и апострофы без необходимости их экранировать.

In [31]:

```
print('''Строка1
Одинарные кавычки '
Двойные кавычки "
Строка2 ''')
```

```
Строка1
Одинарные кавычки '
Двойные кавычки "
Строка2
```

Если перед строкой разместить модификатор `r` (сокращение от raw), то специальные символы внутри строки выводятся как есть. Например, символ `\n` не будет преобразован в символ перевода строки. Иными словами, он будет считаться последовательностью двух символов: `\` и `n`.

In [32]:

```
print('Строка1\nСтрока2')
print(r'Строка1\nСтрока2')
```

```
Строка1
Строка2
Строка1\nСтрока2
```

Если модификатор не указать, то все слэши в пути необходимо экранировать:

In [33]:

```
print('C:\\Python32\\lib\\site-packages')
```

```
C:\Python32\lib\site-packages
```

In [34]:

```
# raw строка удобна для записи путей:
print(r'C:\Python32\lib\site-packages')
```

```
C:\Python32\lib\site-packages
```

In [35]:

```
# проблема: обратный слеш перед кавычкой нужно экранировать
print(r'C:\Python32\lib\site-packages\')
```

```
  File "<ipython-input-35-f88735a4bc25>", line 2
    print(r'C:\Python32\lib\site-packages\')
                                           ^
SyntaxError: EOL while scanning string literal
```

Если в конце строки расположен символ \ ,то его необходимо экранировать, иначе будет выведено сообщение об ошибке:

In [36]:

```
print("string\")
```

```
  File "<ipython-input-36-2b033448615f>", line 1
    print("string\")
                   ^
SyntaxError: EOL while scanning string literal
```

In [37]:

```
print("string\\")
```

```
string\
```

Запись специальных символов из таблицы символов Unicode:

In [38]:

```
euros = "\N{euro sign} \u20AC \U000020AC"
print(euros)
```

```
€ € €
```

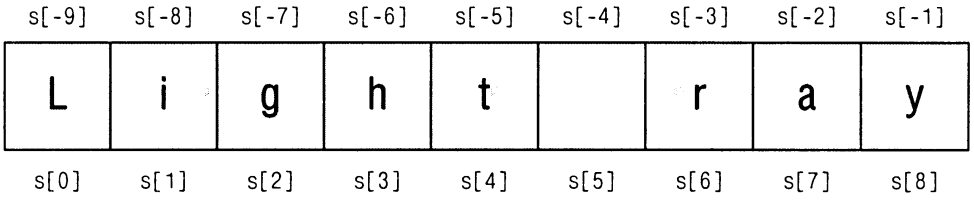
>

Индексирование строк

- к оглавлению

По сути строки являются неизменяемыми последовательностями (массивами) и все функциональные возможности, существующие у неизменяемых последовательностей, могут использоваться и у строк.

Индексирование строк начинается с 0 и до len(s) - 1:



Индексирование строк

In [39]:

```
lr = 'Light ray'
lr
```

Out[39]:

```
'Light ray'
```

In [40]:

```
# индексирование строк начинается с 0:
lr[0]
```

Out[40]:

```
'L'
```

In [41]:

```
len(lr)
```

Out[41]:

```
9
```

In [42]:

```
# последний символ строки имеет индекс на 1 меньший, чем длина строки:
lr[8]
```

Out[42]:

```
'y'
```

In [43]:

```
# выход за пределы последовательности вызывает ошибку:
lr[9]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-43-7e4641780de8> in <module>
      1 # выход за пределы последовательности вызывает ошибку:
----> 2 lr[9]

IndexError: string index out of range
```

In [44]:

```
# Изменять строки нельзя!
lr[1] = 'F'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-cef4942fbd0e> in <module>
      1 # Изменять строки нельзя!
----> 2 lr[1] = 'F'

TypeError: 'str' object does not support item assignment
```

В Python возможно обращение к последовательности с отрицательными индексами:

- -1 - самый последний элемент последовательности
- -2 - второй с конца
- ...
- -len(s) - первый символ последовательности

In [31]:

```
# самый последний элемент последовательности:
lr[-1]
```

Out[31]:

```
'y'
```

In [45]:

```
# запись с отрицательным индексом гораздо удобнее, чем использование len:
lr[len(lr) - 1]
```

Out[45]:

```
'y'
```

In [46]:

```
len(lr) - 1
```

Out[46]:

```
8
```

In [47]:

```
# второй с конца:
lr[-2]
```

Out[47]:

```
'a'
```

In [48]:

```
# первый символ последовательности:
lr[-9]
```

Out[48]:

```
'L'
```

In [49]:

```
# Т.к. отрицательные индексы отсчитваются с -1, а положительные с 0,
# то и модуль крайних положительных и отрицательных индексов отличается на 1:
print(f'Строка: {lr}; первый символ: {lr[-9]}; последний символ: {lr[8]}')
```

```
Строка: Light ray; первый символ: L; последний символ: y
```

In [50]:

```
# выход за пределы последовательности вызывает ошибку:
lr[-10]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-50-974571d4294a> in <module>
      1 # выход за пределы последовательности вызывает ошибку:
----> 2 lr[-10]

IndexError: string index out of range
```

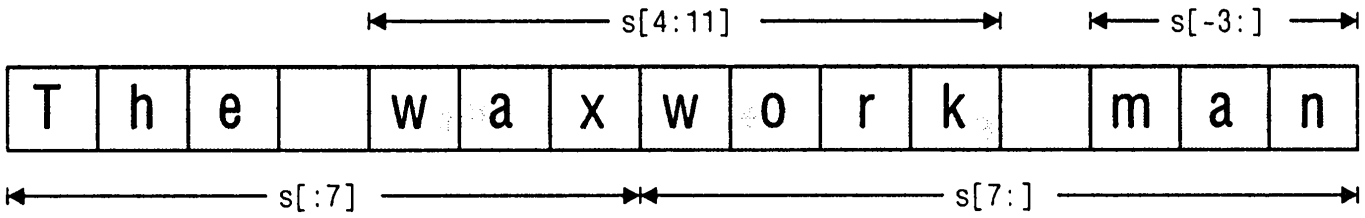
Получение срезов

Три формы оператора получения среза:

- `seq[start]`
- `seq[start:end]`
- `seq[start:end:step]`

Получение среза с 2 параметрами: `seq[start:end]`

Срез возвращает подстроку от индекса `start` до индекса `end` (не включая символ с индексом `end` !)



Срезы строк

In [51]:

```
wm = 'The waxwork man'
wm
```

Out[51]:

```
'The waxwork man'
```

In [52]:

```
# заданы 2 положительных параметра:
wm[4:11]
```

Out[52]:

```
'waxwork'
```

In [53]:

```
wm[:7] # отсутствие первого индекса интерпретируется как значение 0
```

Out[53]:

```
'The wax'
```

In [40]:

```
wm[7:] # отсутствие второго индекса интерпретируется как срез до последнего символа включит
```

Out[40]:

```
'work man'
```

In [41]:

```
wm[:] # от 0-го до последнего символа включительно
```

Out[41]:

```
'The waxwork man'
```

In [54]:

```
wm[4:3] # нет ошибки, вернет пустую строку
```

Out[54]:

```
''
```

In [56]:

```
wm[-3:-1]
```

Out[56]:

```
'ma'
```

In [57]:

```
wm[-3:]
```

Out[57]:

```
'man'
```

In [58]:

```
wm[:-3]
```

Out[58]:

```
'The waxwork '
```

In [59]:

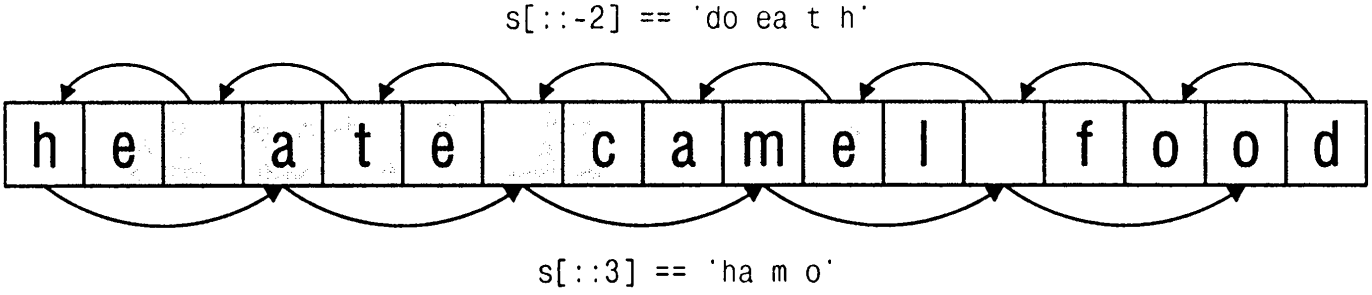
```
wm[-3:3]
```

Out[59]:

```
''
```

Получение среза с 3 параметрами: `seq[start:end:step]`

от символа с индексом `start` до символа с индексом `end` (не включая символ `end` !) с шагом `step`



Срезы строк с шагом

In [60]:

```
acf = 'he ate camel food'
acf
```

Out[60]:

```
'he ate camel food'
```

In [62]:

```
acf[::3]
```

Out[62]:

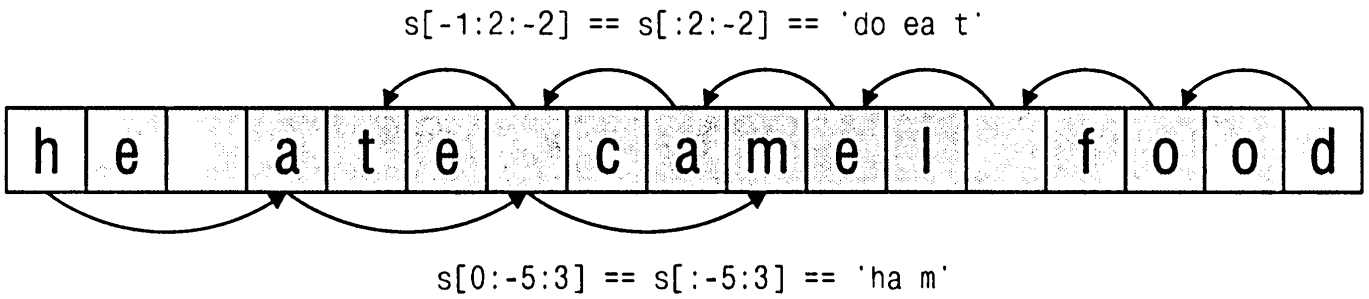
```
'ha m o'
```

In [63]:

```
acf[::-2]
```

Out[63]:

```
'do ea t h'
```



Срезы строк с шагом (2)

In [65]:

```
acf[:2:-2]
```

Out[65]:

```
'do ea t'
```

In [51]:

```
acf[-1:2:-2]
```

Out[51]:

```
'do ea t'
```

In [66]:

```
acf[0:-5:3]
```

Out[66]:

```
'ha m'
```

In [67]:

```
acf[:-5:3]
```

Out[67]:

```
'ha m'
```

In [68]:

```
# можно организовать цикл по символам строки:
for i in acf:
    print(i,  end="-")
```

```
h-e- -a-t-e- -c-a-m-e-l- -f-o-o-d-
```

>

Основные функции для работы со строками

- к оглавлению

Т.к. строки являются неизменяемыми объектами, функции, выполняющие их преобразование, возвращают новые объекты строк.

In [71]:

```
s
```

Out[71]:

```
'Hello world'
```

In [72]:

```
# длина последовательности:
len(s)
```

Out[72]:

```
11
```

Проверка вхождения подстроки в строку

In [73]:

```
s
```

Out[73]:

```
'Hello world'
```

In [75]:

```
# оператор проверки вхождения подстроки в строку:
'e' in s
```

Out[75]:

```
True
```

In [76]:

```
'l' in s
```

Out[76]:

```
True
```

In [77]:

```
'x' in s
```

Out[77]:

```
False
```

In [78]:

```
'L' in s
```

Out[78]:

```
False
```

In [79]:

```
'wor' in s
```

Out[79]:

```
True
```

Операция конкатенации ("склеивания") строк

In [80]:

```
# конкатенация строк при помощи оператора +:
"Строка1" + "Строка2"
```

Out[80]:

```
'Строка1Строка2'
```

In [81]:

```
# конкатенация нескольких строк (неэффективный вариант):
"Строка1" + "Строка2" + "Строка3" + "Строка4"
```

Out[81]:

```
'Строка1Строка2Строка3Строка4'
```

In [82]:

```
s1 = 'Начало'
# добавление в конец последовательности:
s1 += 'Конец' # вместо: s1 = s1 + 'Конец'
s1
```

Out[82]:

```
'НачалоКонец'
```

Повторная конкатенация

In [83]:

```
# повторяем последовательность 3 раза:
s * 3
```

Out[83]:

```
'Hello worldHello worldHello world'
```

In [67]:

```
s1
```

Out[67]:

```
'НачалоКонец'
```

In [84]:

```
# дублирование с присвоением:
s1 *= 4
s1
```

Out[84]:

```
'НачалоКонецНачалоКонецНачалоКонецНачалоКонец'
```

Преобразование строк

`strip([<Символы>])` - удаляет пробельные или указанные символы в начале и в конце строки. Пробельными символами считаются:

- пробел
- символ перевода строки (\n)
- символ возврата каретки (\r)
- символы горизонтальной (\t) и вертикальной (\v) табуляции

In [85]:

```
'  \n\r\v\t strstrstrokstrstrstr \n\r\v '.strip()
```

Out[85]:

```
'strstrstrokstrstrstr'
```

`lstrip( [<Символы>] )` - удаляет пробельные или указанные символы в начале строки

`rstrip( [<Символы>] )` - удаляет пробельные или указанные символы в конце строки

`split ( [<Разделитель> [, <Лимит> ]])` - разделяет строку на подстроки по указанному разделителю и добавляет их в список. Если первый параметр не указан или имеет значение `None`, то в качестве разделителя используется символ пробела. Если во втором параметре задано число, то в списке будет указанное количество подстрок. Если подстрок больше указанного количества, то список будет содержать еше один элемент, в котором будет остаток строки.

In [87]:

```
sw = "word1 \nword2\nword3"
sw.split("\n")
```

Out[87]:

```
['word1 ', 'word2', 'word3']
```

Если в строке содержатся несколько пробелов подряд и разделитель не указан, то пустые элементы не будут добавлены в список. При использовании другого разделителя могут быть пустые элементы.

In [60]:

```
sw2 = 'word    word2  word3  '
sw2.split()
```

Out[60]:

```
['word', 'word2', 'word3']
```

In [88]:

```
sw2 = 'word,,,,word2,,word3,,,'
sw2.split(',')
```

Out[88]:

```
['word', '', '', '', 'word2', '', 'word3', '', '', '']
```

`splitlines([True])` - разделяет строку на подстроки по символу перевода строки (`\n`) и добавляет их в список. Символы новой строки включаются в результат, только если необязательный параметр имеет значение `True` . Если разделитель не найден в строке, то список будет содержать только один элемент.

In [71]:

```
'word1\nword2\nword3'.splitlines()
```

Out[71]:

```
['word1', 'word2', 'word3']
```

In [89]:

```
'word1\nword2\nword3'.splitlines(True)
```

Out[89]:

```
['word1\n', 'word2\n', 'word3']
```

`join()` - преобразует последовательность (в частности, список) в строку. Элементы добавляются через указанный разделитель. Формат метода:

`<Строка> = <Разделитель>.join(<Последовательность>)`

Использование `join()` является лучшим способом "сборки" строки из подстрок, хранящихся в списке или других итерируемых структурах данных.

In [93]:

```
' => '.join(['wordl',  'word2',  'word3'])
```

Out[93]:

```
'wordl => word2 => word3'
```

In [94]:

```
' '.join(['wordl',  'word2',  'word3'])
```

Out[94]:

```
'wordl word2 word3'
```

In [95]:

```
lst = list(s)
lst
```

Out[95]:

```
['H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
```

In [97]:

```
"".join(lst)
```

Out[97]:

```
'Hello world'
```

Для изменения регистра символов предназначены следующие методы:

- `upper()` - заменяет все символы строки соответствующими прописными буквами;
- `lower()` - заменяет все символы строки соответствующими строчными буквами;
- `swapcase()` - заменяет все строчные символы соответствующими прописными буквами, а все прописные символы - строчными;
- `capitalize()` - делает первую букву прописной;
- `title()` - делает первую букву каждого слова прописной.

In [98]:

```
print("строка".upper())
print("СТРОКА".lower())
print("СТРОКА строка".swapcase())
print("строка  строка".capitalize())
st =  "первая  буква  каждого слова  станет  прописной"
print(st.title())
```

```
СТРОКА
строка
строка СТРОКА
Строка  строка
Первая  Буква  Каждого Слова  Станет  Прописной
```

Работа с кодом символа

- функция `chr(<Код  символа>)` - возвращает символ по указанному коду;
- функция `ord(<Символ>)` - возвращает код указанного символа.

In [99]:

```
print(ord("П"))
```

```
1055
```

In [102]:

```
print(chr(1055))
```

```
П
```

Поиск и замена в строке

`find()` - ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Формат метода: `<Строка>.find(<Подстрока>[, <Начало>[, <Конец>]])`

- Если подстрока в строку не входит, то возвращается значение -1.
- Метод зависит от регистра символов.
- Если начальная позиция не указана, то поиск будет осуществляться с начала строки.
- Если параметры `<Начало>` и `<Конец>` указаны, то производится операция извлечения среза `<Строка>[<Начало>:<Конец>]` и поиск подстроки будет выполняться в этом фрагменте.

In [104]:

```
sf = "пример пример Пример"
sf.find("при"), sf.find("При"),  sf.find("тест")
```

Out[104]:

```
(0, 14, -1)
```

In [80]:

```
sf.find("при", 6), sf.find("при", 0, 6), sf.find("при", 7, 12)
```

Out[80]:

```
(7, 0, 7)
```

`index()` - метод аналогичен методу `find()`, но если подстрока в строку не входит, то возбуждается исключение ValueError. Параметры метода: `<Строка>.index(<Подстрока>[, <Начало>[, <Конец>]])`.

In [81]:

```
sf.index("при"), sf.index("При")
```

Out[81]:

```
(0, 14)
```

In [82]:

```
sf.index("тест")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-82-f08eacdc32e6> in <module>()
----> 1 sf.index("тест")

ValueError: substring not found
```

`rfind()` - ищет подстроку в строке. Возвращает позицию **последнего** вхождения подстроки в строку. Формат метода: `<Строка>.rfind(<Подстрока>[, <Начало>[, <Конец>]])` . Аналогичен методу `find()` .

`rindex()` - метод аналогичен методу `index()` , но возвращает позицию **последнего** вхождения. Если подстрока в строку не входит, то возбуждается исключение ValueError. Параметры метода: `<Строка>.rindex(<Подстрока>[, <Начало>[, <Конец>]])` .

In [83]:

```
sf.rfind("при"), sf.rfind("При"),  sf.rfind("тест")
```

Out[83]:

```
(7, 14, -1)
```

In [84]:

```
sf.rindex("при"), sf.rindex("При")
```

Out[84]:

```
(7, 14)
```

`count()` - возвращает число вхождений подстроки в строку. Если подстрока в строку не входит, то возвращается значение 0. Метод зависит от регистра символов. Формат метода: `<Строка>.count(<Подстрока>[, <Начало>[, <Конец>]])` .

In [85]:

```
sf.count("при"),  sf.count("при",  6) , sf.count("При")
```

Out[85]:

```
(2, 1, 1)
```

`startswith()` - проверяет, начинается ли строка с указанной подстроки. Если начинается, то возвращается значение `True` , в противном случае - `False` . Метод зависит от регистра символов. Параметры метода: `<Строка>.startswith(<Подстрока>[, <Начало>[, <Конец>]])` .

Если начальная позиция не указана, сравнение будет производиться с началом строки.

`endswith()` - проверяет, заканчивается ли строка указанной подстрокой. Если заканчивается, то возвращается значение `True` , в противном случае - `False` . Метод зависит от регистра символов.

In [86]:

```
sf2 = "пример  пример Пример ,Пример"
sf2.startswith("при"), sf2.startswith("При")
```

Out[86]:

```
(False, False)
```

In [87]:

```
sf3 = "подстрока  ПОДСТРОКА"
sf3.endswith("ока"), sf3.endswith("ОКА")
```

Out[87]:

```
(False, True)
```

`replace()` - производит замену всех вхождений подстроки в строке на другую подстроку и возвращает результат в виде новой строки. Метод зависит от регистра символов. Параметры метода: `<Строка>.replace(<Подстрока для замены>, <Новая подстрока>[, <Максимальное количество замен>])`

In [2]:

```
sf4 = "Привет, Петя"
print(sf4.replace("Петя", "Вася"))
```

```
Привет, Вася
```

Проверка типа содержимого строки

- `isdigit()` - возвращает True, если строка содержит только цифры, в противном случае - False;
- `isdecimal()` - возвращает True, если строка содержит только десятичные символы, в противном случае- False. Обратите внимание на то, что к десятичным символам относятся не только десятичные цифры в кодировке ASCII, но и надстрочные и подстрочные десятичные цифры в других языках.
- `isnumeric()` - возвращает True, если строка содержит только числовые символы, в противном случае - False. Обратите внимание на то, что к числовым символам относятся не только десятичные цифры в кодировке ASCII, но символы римских чисел, дробные числа и др.
- `isalpha()` - возвращает True, если строка содержит только буквы, в противном случае - False. Если строка пустая, то возвращается значение False.
- `isspace()` - возвращает True, если строка содержит только пробельные символы, в противном случае - False.
- `isalnum()` - возвращает True, если строка содержит только буквы и (или) цифры, в противном случае- False. Если строка пустая, то возвращается значение False.
- `islower()` - возвращает True, если строка содержит буквы, и они все в нижнем регистре, в противном случае - False. Помимо букв строка может иметь другие символы, например цифры.
- `isupper()` - возвращает True, если строка содержит буквы, и они все в верхнем регистре, в противном случае - False. Помимо букв строка может иметь другие символы, например цифры.
- `istitle()` - возвращает True, если строка содержит буквы, и первые буквы всех слов являются заглавными, в противном случае - False. Помимо букв строка может иметь другие символы,

например цифры.

>

Вывод на экран, функция print

- к оглавлению

Формат функции `print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)`

Передает строковые представления объектов `objects` в выходной поток (по умолчанию - стандартный выходной поток).

- `*objects` - объект, направляемый на печать (один объект или несколько объектов, указанных через запятую) Необязательные параметры:
- `file` - выходной поток, в который направляется печать (по умолчанию sys.stdout)
- `sep` - строка, которая вставляется между строковыми представлениями нескольких объектов (по умолчанию - пробел)
- `end` - строка, добавляемая в конце вывода (по умолчанию - `\n`)
- `flush` - форсировать ли сброс (flush) в потоке вывода.

In [105]:

```
print('str1')
```

```
str1
```

In [106]:

```
# конкатенация строкового представления нескольких объектов
# (по умолчанию строки склеиваются с помощью пробелов):
print('str1', 42, 42.2)
```

```
str1 42 42.2
```

In [107]:

```
# склейка с помощью другого символа:
print('str1', 42, 42.2, sep='-')
```

```
str1-42-42.2
```

In [108]:

```
for i in range(10):
    print(i) # по умолчанию в конце вывода добавляется \n
```

```
0
1
2
3
4
5
6
7
8
9
```

In [109]:

```
for i in range(10):
    print(i, end=' ')
```

```
0 1 2 3 4 5 6 7 8 9
```

In [110]:

```
for i in range(10):
    print(i, end='')
```

```
0123456789
```

In [111]:

```
# использование файла в качестве потока вывода:
with open('test.txt', mode='w') as f:
    print('string 1', file=f)
```

>

Современный способ форматирования текста в Python

- к оглавлению

Начиная с версии 3.6 в Python появился новый способ форматирования строк - **f-строки**, которые буквально означают "formatted string". Этот способ форматирования улучшает читаемость кода, а также работает быстрее, чем другие способы форматирования. F-строки задаются с помощью литерала `f` перед кавычками.

In [109]:

```
print("обычная строка")
print(f"f-строка")
```

```
обычная строка
f-строка
```

Способы форматирования строк

В Python существует 5 способов форматирования строк. f-строки - это пятый, самый современный, способ форматирования строк в Python, который очень похож на использование метода format().

0. Конкатенация

Грубый способ форматирования, в котором мы просто склеиваем несколько строк с помощью операции сложения:

In [110]:

```
name = "Петя"
age = 20
print("Меня зовут " + name + ". Мне " + str(age) + " лет.")
```

```
Меня зовут Петя. Мне 20 лет.
```

1. %-форматирование

Широко распространненый старый способ, который перешел в Python из языка C. Передавать значения в строку можно через списки и кортежи, а также с помощью словаря. Во втором случае значения помещаются не по позиции, а в соответствии с именами.

In [112]:

```
name = "Дмитрий"
age = 25
print("Меня зовут %s. Мне %d лет." % (name, age))
print("Меня зовут %(name)s. Мне %(age)d лет." % {"name": name, "age": age})
```

```
Меня зовут Дмитрий. Мне 25 лет.
Меня зовут Дмитрий. Мне 25 лет.
```

2. Template-строки.

Этот способ появился в Python 2.4, как замена %-форматированию, но популярным так и не стал. Поддерживает передачу значений по имени и использует $-синтаксис как в PHP.

In [112]:

```
from string import Template
name = "Дмитрий"
age = 25
s = Template('Меня зовут $name. Мне $age лет.')
print(s.substitute(name=name, age=age))
```

```
Меня зовут Дмитрий. Мне 25 лет.
```

3. Форматирование с помощью метода format().

В Python 3 появился новый способ форматирования - метод format() у строк.

In [113]:

```
name = "Дмитрий"
age = 25
print("Меня зовут {} Мне {} лет.".format(name, age))
```

```
Меня зовут Дмитрий Мне 25 лет.
```

4. f-строки.

f-строки использовать значения переменных, которые есть в текущей области видимости, и подставлять их в строку. В самой строке вам лишь нужно указать имя этой переменной в фигурных скобках.

Форматирование с помощью f-строк появилось в Python 3.6 . Этот способ похож на форматирование с помощью метода format(), но гибче, читабельней и быстрей.

In [113]:

```
name = "Дмитрий"
age = 25
print(f"Меня зовут {name} Мне {age} лет.")
```

```
Меня зовут Дмитрий Мне 25 лет.
```

f-строки также поддерживают расширенное форматирование чисел:

In [115]:

```
from math import pi
print(f"Значение числа pi: {pi:.2f}")
```

```
Значение числа pi: 3.14
```

С помощью f-строк можно форматировать дату без вызова метода strftime():

In [114]:

```
from datetime import datetime as dt
now = dt.now()
print(f"Текущее время {now:%d.%m.%Y %H:%M}")
```

```
Текущее время 27.07.2021 15:21
```

Они поддерживают базовые арифметические операции прямо в строках:

In [115]:

```
x = 10
y = 5
print(f"{x} x {y} / 2 = {x * y / 2}")
```

```
10 x 5 / 2 = 25.0
```

f-строки позволяют обращаться к значениям списков по индексу:

In [116]:

```
planets = ["Меркурий", "Венера", "Земля", "Марс"]
print(f"Мы живем на планете {planets[2]}")
```

```
Мы живем на планете Земля
```

А также к элементам словаря по ключу:

In [117]:

```
planet = {"name": "Земля", "radius": 6378000}
print(f"Планета {planet['name']}. Радиус {planet['radius']/1000} км.")
```

```
Планета Земля. Радиус 6378.0 км.
```

Можно использовать как строковые, так и числовые ключи. Точно также, как в обычном Python коде:

In [120]:

```
digits = {0: 'ноль', 'one': 'один'}
print(f"0 - {digits[0]}, 1 - {digits['one']}")
```

```
0 - ноль, 1 - один
```

В f-строках можно вызывать методы объектов:

In [118]:

```
name = "Дмитрий"
print(f"Имя: {name.upper()}")
```

```
Имя: ДМИТРИЙ
```

А также вызывать функции:

In [119]:

```
print(f"13 / 3 = {round(13/3)}")
```

```
13 / 3 = 4
```

Кроме удобства и большой гибкости f-строки являются и одним из самых производительных (быстрых) способов форматирования строк на Python.

>

Расширенное форматирование

- к оглавлению

Пример. Сформировать строку форматирования, которая приводит к формированию следующей таблицы:

```
    01234567890123456789012345678901234567890123456789
    |   1 | xxx59.06|      453 | 00000001 |
    |   5 | xx159.00|  123,453 | 00000111 |
    |  15 | x-159.10| -     12 | 00010000 |
    | 105 | -1059.10|    1,200 | 11111111 |
```

In [123]:

```
print('0123456789'*4)

def f(a,b,c,d):
    s2 = f'|{a: >3}|{b:x>8.2f}|{c:=8}|{d:0>8}|' # .format(a,b,c,d)
#     s2='|{: >3}|{:#>8.2f}|{:=8}|{:0>8}|'.format(a,b,c,d)
    return s2

print(f(1, 59.06, 453, 1))
print(f(5, 159.00, 123.453, 111))
print(f(15, -159.10, -12,10000))
print(f(105, -1059.10, 1200, 1111111))
```

```
01234567890123456789012345678901234567890123456789
|  1|xxx59.06|     453|00000001|
|  5|xx159.00| 123.453|00000111|
| 15|x-159.10|-     12|00010000|
|105|-1059.10|    1200|01111111|
```

Подробная информация по возможностям расширенного форматирования:

http://www.python-course.eu/python3_formatted_output.php (http://www.python-course.eu/python3_formatted_output.php)

Возможности расширенного форматирования в f-строках и у функции format() совпадают. Синтаксис функции format(): `<Строка специального формата>.format(*args, **kwargs)`

В параметре `<Формат>` указывается значение, имеющее следующий синтаксис:

`[ [<Заполнитель>] <Выравнивание>] [<Знак>] [ #] [0] [<Ширина>] [,] [.<Точность>] [ <Преобразование>]`

In [136]:

```
i = 3
f"{i:10}" #  10 - это ширина  поля
```

Out[136]:

```
'         3'
```

По умолчанию значение внутри поля выравнивается по правому краю. Управлять выравниванием позволяет параметр <Выравнивание>. Можно указать следующие значения:

- < - по левому краю;
- > - по правому краю;
- ^ - по центру поля;
- = - знак числа выравнивается по левому краю, а число по правому краю.

In [137]:

```
"'{0:<10}' '{1:>10}' '{2:^10}' '{3:=10}'".format(3,  3,  3, -3)
```

Out[137]:

```
"'3         ' '         3' '    3     ' '-        3'"
```

Пространство между знаком и числом по умолчанию заполняется пробелами, а знак положительного числа не указывается. Чтобы вместо пробелов пространство заполнялось нулями, необходимо указать нуль перед шириной поля. Такого же эффекта можно достичь, указав нуль в параметре <Заполнитель>. В этом параметре допускаются и другие символы, которые будут выводиться вместо пробелов:

In [69]:

```
"'{0:=010}'  '{1:=010}'".format(-3,  3)
```

Out[69]:

```
"'-000000003'  '0000000003'"
```

In [138]:

```
"'{0:0=10}'  '{1:_=10}'".format(-3,  3)
```

Out[138]:

```
"'-000000003'  '_________3'"
```

In [139]:

```
"'{0:*<10}' '{1:+>10}' '{2:.^10}'".format(3, 3, 3)
```

Out[139]:

```
"'3*********' '+++++++++3' '....3.....'"
```

Управлять выводом знака числа позволяет параметр <Знак>. Допустимые значения:

- + - задает обязательный вывод знака как для отрицательных, так и для положительных чисел;
- - -вывод знака только для отрицательных чисел (значение по умолчанию);
- пробел - вставляет пробел перед положительным числом. Перед отрицательным числом будет стоять минус.

In [73]:

```
"'{0:+}' '{1:+}' '{0:-}' '{1:-}'".format(3,  -3)
```

Out[73]:

```
"'+3' '-3' '3' '-3'"
```

In [76]:

```
"'{0: }' '{1: }'".format(3,  -3)
```

Out[76]:

```
"' 3' '-3'"
```

Для целых чисел в параметре <Преобразование> могут быть указаны следующие опции:

- b - двоичное значение:
- c - преобразует целое число в соответствующий символ;
- d - десятичное значение;
- n - аналогично опции d, но учитывает настройки локали. Например, выведем большое число с разделением тысячных разрядов пробелом;
- o - восьмеричное значение;
- x - шестнадцатеричное значение в нижнем регистре;
- X - шестнадцатеричное значение в верхнем регистре.

In [140]:

```
"'{0:b}' '{0:#b}'".format(3)
```

Out[140]:

```
"'11' '0b11'"
```

In [141]:

```
"'{0:c}'".format(10)
```

Out[141]:

```
"'\n'"
```

In [142]:

```
"'{0:d}'".format(100)
```

Out[142]:

```
"'100'"
```

In [145]:

```
import locale
locale.setlocale(locale.LC_NUMERIC, 'Russian_Russia.1251')
```

Out[145]:

```
'Russian_Russia.1251'
```

In [147]:

```
print("{0:n}".format(100000000))
```

```
100 000 000
```

In [146]:

```
print("{0:,d}".format(100000000))
```

```
100,000,000
```

In [95]:

```
"'{0:o}' '{0:x}' '{0:X}'".format(200)
```

Out[95]:

```
"'310' 'c8' 'C8'"
```

Для вещественных чисел в параметре <Преобразование> могут быть указаны следующие опции:

- f и F - вещественное число в десятичном представлении. Задать количество знаков после запятой позволяет параметр <Точность>;
- e - вещественное число в экспоненциальной форме (буква "e" в нижнем регистре);

- E - вещественное число в экспоненциальной форме (буква "e" в верхнем регистре);
- n - аналогично опции g, но учитывает настройки локали;
- % - умножает число на 100 и добавляет символ процента в конец. Значение отображается в соответствии с опцией f.

In [106]:

```
"'{0:f}' '{1:f}' '{2:f}'".format(30, 18.6578145, -2.5)
```

Out[106]:

```
"'30.000000' '18.657815' '-2.500000'"
```

In [107]:

```
"'{0:.7f}' '{1:.2f}'".format(18.6578145, -2.5)
```

Out[107]:

```
"'18.6578145' '-2.50'"
```

In [104]:

```
"'{0:e}' '{1:e}'".format(3000, 18657.81452)
```

Out[104]:

```
"'3.000000e+03' '1.865781e+04'"
```

In [105]:

```
"'{0:E}' '{1:E}'".format(3000, 18657.81452)
```

Out[105]:

```
"'3.000000E+03' '1.865781E+04'"
```

In [148]:

```
"'{0:%}' '{1:.4%}'".format(0.086578, 0.000086578)
```

Out[148]:

```
"'8.657800%' '0.0087%'"
```

In [ ]:

```

```