СПРАВОЧНИК ШКОЛЬНИКА

по языку программирования Python

Составитель: А.Г. Гильдин

СПИСКИ в языке программирования Python

Примеры интерактивной работы со списками

```
>>> s = []  # Пустой список
>>> l = ['s', 'p', ['isok'], 2]
>>> del l[1] # Удалили элемент с индексом 1
>>> s
[]
>>> l
['s', ['isok'], 2]
>>> list('список')
['с', 'п', 'и', 'с', 'о', 'к']
```

Таблица методов работы со списками. В таблице список обозначен словом li.

Метод или команда	Что делает
li.append(x)	Добавляет элемент в конец списка
li.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
li.insert(i, x)	Вставляет на i-ый элемент значение x
li.remove(x)	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
li.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
li.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
li.count(x)	Возвращает количество элементов со значением x
li.sort([key=функция])	Сортирует список на основе функции
li.reverse()	Разворачивает список
li.copy()	Поверхностная копия списка
li.clear()	Очищает список
len(li)	Возвращает количество элементов списка.

Массивы в языке программирования Python

Примеры интерактивной работы с массивами

```
>>> import array
>>> m = array.array('i' ,[0, 9, 0, 0, 7, 0])
>>> m[1]
9
```

Таблица символов, использующихся для указания типа элементов массива

Б.	Тип данных (C++)	байт	Б.	Тип данных (C++)	байт
'b'	signed char	1	'd'	double	8
'h'	signed short	2	'f'	float	4
'H'	unsigned short	2	'l'	signed long	4
'q'	signed long long	8	'I'	unsigned int	2
'B'	unsigned char	1	'Q'	unsigned long long	8
'i'	signed int	2	'L'	unsigned long	4

Таблица методов работы с массивами. В таблице массив обозначен словом a.

Метод или команда	Что делает
a.itemsize	размер в байтах одного элемента в массиве
a.append(x)	добавление элемента в конец массива
a.count(x)	возвращает количество вхождений x в массив
a.extend(iter)	добавление элементов из объекта iter в массив
a.fromfile(F, N)	читает N элементов из файла и добавляет их в конец массива. Файл должен быть открыт на бинарное чтение. Если доступно меньше N элементов, генерируется исключение EOFError, но элементы, которые были доступны, добавляются в массив.
a.fromlist(список)	добавление элементов из списка
a.index(x)	номер первого вхождения x в массив
a.insert(n, x)	вставка элемента x в массив a перед номером n. Отрицательные значения рассматриваются относительно конца массива
a.pop(i)	удаляет i-ый элемент из массива и возвращает его. Если не указать i, то удаляется последний элемент
a.remove(x)	-удалить первое вхождение x из массива

Метод или команда	Что делает
a.reverse()	Сделать обратный порядок элементов в массиве
len(li)	Возвращает количество элементов в массиве
a.tofile(f)	Запись массива в открытый файл
a.tolist()	Возвращает элементы массива в виде списка

СЛОВАРИ в языке программирования Python

Примеры

```
>>> d = {} #пустой словарь
>>> d = {'dict': 1, 'dictionary': 2} #словарь с двумя парами
>>> d = dict.fromkeys(['a', 'b'])# словарь с двумя ключами без
значений
>>> d = dict.fromkeys(['a', 'b'], 100) #у этих двух ключей
значения 100
>>> d = {a: a ** 2 for a in range(7)} #генератор словарей
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
>>> d = {1: 2, 2: 4, 3: 9}
>>> d[1]
2
>>> d[4] = 4 ** 2
>>> d
{1: 2, 2: 4, 3: 9, 4: 16}
```

Таблица методов работы со словарями. В таблице словарь обозначен словом di.

Метод или команда	Что делает
di.values()	возвращает значения в словаре
di.clear()	очищает словарь
di.copy()	возвращает копию словаря
di.get(key[, default])	возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None)
di.keys()	возвращает ключи в словаре
di.items()	возвращает пары (ключ, значение)
di.setdefault(key[, default])	возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ с значением default (по умолчанию None)
len(di)	внешняя функция возвращает количество элементов в словаре
dict.fromkeys(seq[, value])	Это метод класса. Создает словарь с ключами из seq и значением value (по умолчанию None)
di.pop(key[, default])	- удаляет ключ и возвращает значение. Если ключа нет, возвращает default. Если не указан default, возникает исключение
di.popitem()	удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены
di.update([other])	обновляет словарь, добавляя пары (ключ, значение) из other.

Метод или команда	Что делает
	Существующие ключи перезаписываются.

Множества в языке программирования Python

Примеры

```
>>> a = set()
>>> a
set()
>>> a = set('hello')
>>> a
{'h', 'o', 'l', 'e'}
>>> a = {'a', 'b', 'c', 'd'}
>>> a
{'b', 'c', 'a', 'd'}
>>> a = {i ** 2 for i in range(10)} # генератор множеств
>>> a
{0, 1, 4, 81, 64, 9, 16, 49, 25, 36}
>>> a = {}  # А так нельзя!
>>> type(a)
<class 'dict'>
>>> words = ['hello', 'daddy', 'hello', 'mum']
>>> set(words)
{'hello', 'daddy', 'mum'}
```

frozenset – еще один тип данных – множество. Единственное отличие set от frozenset заключается в том, что set - изменяемый тип данных, а frozenset - нет.

Таблица методов работы с множествами. В таблице множество обозначено словом set.

Метод или команда	Что делает
len(s)	количество элементов во множестве s.
in	x in s - принадлежит ли x множеству s.
set.isdisjoint(other)	метод возвращает истину, если множества set и other не имеют общих элементов.
==	set == other - все элементы set принадлежат other, все элементы other принадлежат set.
set.issubset(other) или set <= other	метод возвращает истину, если все элементы множества set содержатся также в множестве other.
set.issuperset(other) или set >= other	метод возвращает истину, если все элементы множества set содержатся также в множестве other.
set.union(other, ...) или set | other | ...	объединение нескольких множеств. Метод возвращает новое множество.
set.intersection(other, ...) или set & other & ...	пересечение нескольких множеств. Метод возвращает новое множество.

Метод или команда	Что делает
set.difference(other, ...) или set - other - ...	возвращает множество из всех элементов set за исключением элементов, принадлежащих также множеству other.
set.symmetric_difference (other) или set ^ other	возвращает множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
set.copy()	метод возвращает копию множества.
	методы, изменяющие множества
set.update(other, ...) или set |= other | ...	объединение множеств.
set.intersection_update(other, ...) или set &= other & ...	пересечение множеств.
set.difference_update(other, ...) или set -= other | ...	разность множеств.
set.symmetric_difference_update(other) или set ^= other	множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
set.add(elem)	добавляет элемент elem в множество.
set.remove(elem)	удаляет элемент elem из множества. Генерирует KeyError, если такого элемента не существует.
set.discard(elem)	удаляет элемент elem, если он находится в множестве.
set.clear()	очистка множества.

СТРОКИ в языке программирования Python

Таблица функций и методов работы со строками. В таблице строка обозначена S

Метод или команда	Что делает
S = 'str'; S = "str"; S = '''str'''; S = """str"""	Литералы строк

Метод или команда	Что делает
S = "s\np\ta\nbbb"	Служебные символы (Экранированные последовательности) например, \n означает перевод строки
S = r"C:\temp\new"	Неформатированные строки (подавляют экранирование)
S = b"byte"	Строка байтов
S1 + S2	Конкатенация (сложение строк)
S1 * 3	Повторение строки
S[i]	Обращение к символу строки с номером i. Первый символ строки имеет индекс 0.
S[i:j:step]	Извлечение среза
len(S)	Длина строки
S.find(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.rfind(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1
S.index(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
S.replace(фрагмент1, фрагмент2,[count])	Заменяет в строке не более count фрагментов1 на фрагменты2. Если count не указан, заменяет все фрагменты.
S.split(символ)	Разбиение строки по разделителю. Возвращает список
S.isdigit()	Состоит ли строка из цифр
S.isalpha()	Состоит ли строка из букв
S.isalnum()	Состоит ли строка из цифр или букв
S.islower()	Состоит ли строка из символов в нижнем регистре
S.isupper()	Состоит ли строка из символов в верхнем регистре
S.isspace()	Состоит ли строка из неотображаемых символов (пробел, символ перевода страницы ('\f'), "новая строка" ('\n'), "перевод каретки" ('\r'), "горизонтальная табуляция" ('\t') и "вертикальная табуляция" ('\v'))
S.istitle()	Начинаются ли слова в строке с заглавной буквы
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру

Метод или команда	Что делает
S.startswith(str)	Начинается ли строка S с фрагмента str. Можно добавлять еще два параметра (beg, end): с какого символа искать и до какого.
S.endswith(str)	Заканчивается ли строка S фрагментом str. Можно добавлять еще два параметра (beg, end): с какого символа искать и до какого.
S.join(список)	Возвращает строку, собирая ее из списка. Между элементами списка вставляется разделитель равный строке S.
ord(символ)	Возвращает ASCII код символа
chr(число)	Возвращает символ по его ASCII коду
S.capitalize()	Возвращает строку, в которой первый символ в верхнем регистре, а все остальные в нижнем
S.center(width, [fill])	Возвращает отцентрованную строку, по краям которой стоит символ fill (пробел по умолчанию)
S.count(str, [start],[end])	Возвращает количество непересекающихся вхождений подстроки в диапазоне [начало, конец] (0 и длина строки по умолчанию)
S.expandtabs([tabsize])	Возвращает копию строки, в которой все символы табуляции заменяются одним или несколькими пробелами, в зависимости от текущего столбца. Если TabSize не указан, размер табуляции полагается равным 8 пробелам
S.lstrip([chars])	Возвращает строку без пробельных символов в начале строки
S.rstrip([chars])	Возвращает строку без пробельных символов в конце строки
S.strip([chars])	Возвращает строку без пробельных символов в начале и в конце строки
S.partition(шаблон)	Возвращает кортеж, содержащий часть перед первым шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий саму строку, а затем две пустых строки
S.rpartition(sep)	Возвращает кортеж, содержащий часть перед последним шаблоном, сам шаблон, и часть после шаблона. Если шаблон не найден, возвращается кортеж, содержащий две пустых строки, а затем саму строку
S.swapcase()	Возвращает строку, в которой переведены символы нижнего регистра в верхний, а верхнего – в нижний
S.title()	Возвращает строку, в которой первая буква каждого слова переведена в верхний регистр, а все остальные в нижний
S.zfill(width)	Возвращает строку, в которой делает длину строки не меньшей width, по необходимости заполняя первые символы нулями
S.ljust(width, fillchar=" ")	Возвращает строку, в которой делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar
S.rjust(width, fillchar=" ")	Возвращает строку, в которой делает длину строки не меньшей width, по

Метод или команда	Что делает
	необходимости заполняя первые символы символом fillchar
S.format(*args, **kwargs)	Форматирование строки. Подробнее здесь: Форматирование строки
int(s)	Пример использования преобразования типов данных. Возвращает число, записанное в строке s.

Форматирование строк с помощью метода format

Если для подстановки требуется только один аргумент, то значение - сам аргумент:

```
>>> 'Hello, {}!'.format('Vasya')
'Hello, Vasya!'
```

А если несколько, то значениями будут являться все аргументы со строками подстановки (обычных или именованных):

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc')
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad')
'abracadabra'
>>> 'Coordinates: {latitude},
{longitude}'.format(latitude='37.24N', longitude='-115.81W')
'Coordinates: 37.24N, -115.81W'
>>> coord = {'latitude': '37.24N', 'longitude': '-115.81W'}
>>> 'Coordinates: {latitude}, {longitude}'.format(**coord)
'Coordinates: 37.24N, -115.81W'
```

Общий синтаксис:

```
поле замены     ::=  "{" [имя поля] ["!" преобразование] [":"
спецификация] "}"

имя поля        ::=  arg_name ("." имя атрибута | "[" индекс "]")*

преобразование  ::=  "r" (внутреннее представление) | "s" (человеческое
представление)

спецификация    ::=  см. ниже
```

Например:

```
>>>
>>> "Units destroyed: {players[0]}".format(players = [1, 2,
3])
'Units destroyed: 1'
>>> "Units destroyed: {players[0]!r}".format(players = ['1',
'2', '3'])
"Units destroyed: '1'"
```

Теперь спецификация формата:

```
спецификация ::=  [[fill]align][sign][#][0][width][,][.precision][type]

заполнитель  ::=  символ кроме '{' или '}'

выравнивание ::=  "<" | ">" | "=" | "^"

знак         ::=  "+" | "-" | " "

ширина       ::=  integer

точность     ::=  integer

тип          ::=  "b" | "c" | "d" | "e" | "E" | "f" | "F" | "g" | "G" |

                  "n" | "o" | "s" | "x" | "X" | "%"
```

Выравнивание производится при помощи символа-заполнителя. Доступны следующие варианты выравнивания:

Флаг	**Значение**
'<'	Символы-заполнители будут справа (выравнивание объекта по левому краю) (по умолчанию).
'>'	выравнивание объекта по правому краю.
'='	Заполнитель будет после знака, но перед цифрами. Работает только с числовыми типами.

'^'	Выравнивание по центру.

Опция "знак" используется только для чисел и может принимать следующие значения:

Флаг	**Значение**
'+'	Знак должен быть использован для всех чисел.
'-'	'-' для отрицательных, ничего для положительных.
'Пробел'	'-' для отрицательных, пробел для положительных.

Поле "тип" может принимать следующие значения:

Тип	**Значение**
'd', 'i', 'u'	Десятичное число.
'o'	Число в восьмеричной системе счисления.
'x'	Число в шестнадцатеричной системе счисления (буквы в нижнем регистре).
'X'	Число в шестнадцатеричной системе счисления (буквы в верхнем регистре).

'e'	Число с плавающей точкой с экспонентой (экспонента в нижнем регистре).
'E'	Число с плавающей точкой с экспонентой (экспонента в верхнем регистре).
'f', 'F'	Число с плавающей точкой (обычный формат).
'g'	Число с плавающей точкой. с экспонентой (экспонента в нижнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'G'	Число с плавающей точкой. с экспонентой (экспонента в верхнем регистре), если она меньше, чем -4 или точности, иначе обычный формат.
'c'	Символ (строка из одного символа или число - код символа).
's'	Строка.
'%'	Число умножается на 100, отображается число с плавающей точкой, а за ним знак %.

Примеры:

```
>>> coord = (3, 5)
>>> 'X: {0[0]};  Y: {0[1]}'.format(coord)
'X: 3;  Y: 5'
>>> "repr() shows quotes: {!r}; str() doesn't:
{!s}".format('test1', 'test2')
"repr() shows quotes: 'test1'; str() doesn't: test2"
>>> '{:<30}'.format('left aligned')
'left aligned                  '
>>> '{:>30}'.format('right aligned')
'                 right aligned'
>>> '{:^30}'.format('centered')
```

```
'           centered           '
>>> '{:*^30}'.format('centered')  # use '*' as a fill char
'**********centered**********'
>>> '{:+f}; {:+f}'.format(3.14, -3.14)  # show it always
'+3.140000; -3.140000'
>>> '{: f}; {: f}'.format(3.14, -3.14)  # show a space for
positive numbers
' 3.140000; -3.140000'
>>> '{:-f}; {:-f}'.format(3.14, -3.14)  # show only the minus
-- same as '{:f}; {:f}'
'3.140000; -3.140000'
>>> # format also supports binary numbers
>>> "int: {0:d};  hex: {0:x};  oct: {0:o};  bin:
{0:b}".format(42)
'int: 42;  hex: 2a;  oct: 52;  bin: 101010'
>>> # with 0x, 0o, or 0b as prefix:
>>> "int: {0:d};  hex: {0:#x};  oct: {0:#o};  bin:
{0:#b}".format(42)
'int: 42;  hex: 0x2a;  oct: 0o52;  bin: 0b101010'
>>> points = 19.5
>>> total = 22
>>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 88.64%'
```

f - строки

Общий синтаксис: поставьте перед строкой f, а в самой строке переменные или выражения заключайте в фигурные скобки.

```
>>> name = "Eric"
>>> age = 74
>>> f"Hello, {name}. You are {age}."
'Hello, Eric. You are 74.
```

Вы можете использовать многострочные строки:

```
>>> name = "Eric"
>>> profession = "comedian"
>>> affiliation = "Monty Python"
>>> message = (
... f"Hi {name}. "
... f"You are a {profession}. "
... f"You were in {affiliation}."
... )
>>> message
```

```
'Hi Eric. You are a comedian. You were in Monty Python.'
```

Если вы хотите распределить строки по нескольким строкам, вы можете так же сделать это с помощью \:

```
>>> message = f"Hi {name}. " \
... f"You are a {profession}. " \
... f"You were in {affiliation}."
...
>>> message
'Hi Eric. You are a comedian. You were in Monty Python.'
```

Или с помощью тройных кавычек:

```
>>> message = f"""
... Hi {name}.
... You are a {profession}.
... You were in {affiliation}.
... """
...
>>> message
'\n Hi Eric.\n You are a comedian.\n You were in Monty Python.\n'
```

Примеры из официальной документации (https://docs.python.org/3/reference/lexical_analysis.html#f-strings)

```
>>> name = "Fred"
>>> f"He said his name is {name!r}."
"He said his name is 'Fred'."
>>> f"He said his name is {repr(name)}."  # repr() is equivalent to !r
"He said his name is 'Fred'."
>>> width = 10
>>> precision = 4
>>> value = decimal.Decimal("12.34567")
>>> f"result: {value:{width}.{precision}}"  # nested fields
'result:      12.35'
>>> today = datetime(year=2017, month=1, day=27)
>>> f"{today:%B %d, %Y}"  # using date format specifier
'January 27, 2017'
>>> f"{today=:%B %d, %Y}" # using date format specifier and debugging
'today=January 27, 2017'
>>> number = 1024
>>> f"{number:#0x}"  # using integer format specifier
'0x400'
```

```
>>> foo = "bar"
>>> f"{ foo = }" # preserves whitespace
" foo = 'bar'"
>>> line = "The mill's closed"
>>> f"{line = }"
'line = "The mill\'s closed"'
>>> f"{line = :20}"
"line = The mill's closed   "
>>> f"{line = !r:20}"
'line = "The mill\'s closed" '
```

Регулярные выражения

```
import re
```

Регулярные выражения имеют спецсимволы, которые нужно экранировать. Вот их список: . ^ $ * + ? { } [ ] \ | ( ). Экранирование осуществляется обычным способом — добавлением \ перед спецсимволом. Кроме этого, экранировать нужно символ «-», он используется для задания диапазонов.

Таблица методов работы с регулярными выражениями.

Метод или команда	Что делает
re.match(pattern, string)	Ищет строку, соответствующую шаблону pattern именно с нулевого символа строки string. Возвращает объект, у которого есть численные методы start() и end(). Если не нашел - None. Можно использовать в качестве проверки соответствия строки string шаблону pattern, используя функцию re.math в обычной условной конструкции.
re.search(pattern, string)	Ищет первое вхождение строки, соответствующей шаблону pattern слева направо в строке string. Возвращает объект, у которого есть численные методы start() и end(). Если не нашел - None.
re.findall(pattern, string)	Ищет все вхождения.
re.split(pattern, string, [maxsplit=0])	Разбивает строку string, используя в качестве разделителя шаблон pattern. Если задан параметр maxsplit, то выполняется не более заданного количества первых слева направо разбиений. Возвращает список строк.
re.sub(pattern, repl, string)	Возвращает строку в которой все подстроки строки string, соответствующие шаблону pattern заменены строкой repl.
re.compile(pattern, repl, string)	

Таблица операторов для формирования шаблонов. Напомним, что r перед строкой отключает экранирование внутри этой строки. Таким образом, перед шаблоном скорее всего, удобно поставить символ r.

Оператор	Что делает
.	Один любой символ, кроме новой строки \n.
?	0 или 1 вхождение шаблона слева
+	1 и более вхождений шаблона, записанного слева от знака +, или к круглым скобкам (здесь и далее)
*	0 и более вхождений шаблона, записанного слева от знака *
\w	Любая цифра или буква (\W — все, кроме буквы или цифры)
\d	Любая цифра [0-9] (\D — все, кроме цифры)
\s	Любой пробельный символ (\S — любой непробельный символ)
\b	Граница слова (\B – не граница слова)
[..]	Один из символов в скобках ([^..] — любой символ, кроме тех, что в скобках)
\	Экранирование специальных символов (\. означает точку или \+ — знак «плюс»)
^ и $	Начало и конец строки соответственно
{n,m}	От n до m вхождений шаблона, указанного перед фигурными скобками ({,m} — от 0 до m; {m,} — не менее m)
{n}	Ровно n вхождений шаблона, указанного перед фигурными скобками
a|b	Соответствует a или b
()	Группирует выражение и возвращает найденный текст
\t, \n, \r	Символ табуляции, новой строки и возврата каретки соответственно

Список всех непробельных символов

```
result = re.findall(r'\w', 'AV is largest Analytics community of India')
```

```
Результат: ['A', 'V', 'i', 's', 'l', 'a', 'r', 'g', 'e', 's', 't', 'A',
'n', 'a', 'l', 'y', 't', 'i', 'c', 's', 'c', 'o', 'm', 'm', 'u', 'n',
'i', 't', 'y', 'o', 'f', 'I', 'n', 'd', 'i', 'a']
```

[а-яА-ЯёЁ] этот шаблон означает все буквы русского языка

Список всех слов

```
result = re.findall(r'\w*', 'AV is largest Analytics community of
India')
```

Результат: `['AV', '', 'is', '', 'largest', '', 'Analytics', '', 'community', '', 'of', '', 'India', '']`

В результат попали пробелы, так как * -это ноль или более подходящих. Если заменить на +, то будут только слова.

```
result = re.findall(r'@\w+.(\w+)', 'abc.test@gmail.com, xyz@test.in, test.first@analyticsvidhya.com, first.test@rest.biz')
```

Результат: `['com', 'in', 'com', 'biz']`

Сторонние библиотеки
Репозиторий питона https://pypi.org/

Модуль math
Import math
Что есть в модуле? dir(math)
методы вызываются через точку после названия модуля

gcd(a,b)	возвращает НОД(a,b)

Модуль datetime
Import datetime
Что есть в модуле? dir(datetime)
методы вызываются через точку после названия модуля

`datetime.now()`	
`datetime.now().time()`	
`datetime.now().date()`	
`date.today()`	Текущая дата
`date.today().weekday()`	Текущий день недели. Нумерация: 0 - воскресенье
`timedelta(seconds=10, weeks=2)`	Класс для создания интервала времени. Отрицательные числа допускаются.

`strftime(строка формата)`	Функция возвращает форматированную строку
`date(2019, 11, 25)`	Пример создания даты

Команда	Значение	Пример
%a	Аббревиатура дня недели	Sun, Mon, ..., Sat (en_US);So, Mo, ..., Sa (de_DE)
%A	Полное название дня недели	Sunday, Monday, ..., Saturday (en_US);Sonntag, Montag, ..., Samstag (de_DE)
%w	День недели как десятичное число, где 0 — воскресенье, а 6 — суббота	0, 1, ..., 6
%d	День месяца в формате из двух цифр	01, 02, ..., 31
%b	Аббревиатура месяца	Jan, Feb, ..., Dec (en_US);Jan, Feb, ..., Dez (de_DE)
%B	Полное название месяца	January, February, ..., December (en_US);Januar, Februar, ..., Dezember (de_DE)
%m	Номер месяца в формате из двух цифр	01, 02, ..., 12
%y	Последние 2 цифры года (год без века)	00, 01, ..., 99
%Y	Год полностью	0001, 0002, ..., 2013, 2014, ..., 9998, 9999
%H	Час в 24-часовом формате из двух цифр	00, 01, ..., 23
%I	Час в 12-часовом формате из двух цифр	01, 02, ..., 12
%p	AM или PM	AM, PM (en_US);am, pm (de_DE)
%M	Минута в формате из двух цифр	00, 01, ..., 59
%S	Секунда в формате их двух цифр	00, 01, ..., 59
%f	Микросекунды в формате из 6 цифр	000000, 000001, ..., 999999
%j	День в году в формате из 3 цифр	001, 002, ..., 366
%U	Неделя в году	00, 01, ..., 53
%c	Принятое, согласно локальным настройкам, представление даты и времени	Tue Aug 16 21:30:00 1988 (en_US);Di 16 Aug 21:30:00 1988 (de_DE)

Команда	Значение	Пример
%x	Принятое, согласно локальным настройкам, представление даты	08/16/88 (None);08/16/1988 (en_US);16.08.1988 (de_DE)
%X	Принятое, согласно локальным настройкам, представление времени	21:30:00 (en_US);21:30:00 (de_DE)
%%	Знак '%'	%

```
Статические методы классов @staticmethod
```

Материалы этого справочника подготовлены с использованием материалов сайтов: https://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html https://pythontutor.ru/lessons/lists/

Потоковый ввод

```
import sys
for line in sys.stdin:
    # rstrip('\n') "отрезает" от строки line идущий справа символ
    # перевода строки, ведь print сам переводит строку
    print(line.rstrip('\n'))

stdin – итератор, следовательно он «пустеет» при проходе по нему.

data = list(map(str.strip, sys.stdin))
```

collections.Counter подсчет количества вхождений

комплект декораторов functools

```
import functools
@functools.lru_cache(maxsize=128)
def fibonacci(n):
if n == 0:
return 0
elif n == 1:
return 1
return fibonacci(n - 1) + fibonacci(n - 2)
```

Генерация псевдослучайных чисел в языке программирования Python (модуль random)

Модуль random позволяет генерировать случайные числа. Прежде чем использовать модуль, необходимо подключить его с помощью инструкции:

```
import random
```

Метод или команда	Что делает
random.random()	возвращает псевдослучайное число от 0.0 до 1.0
random.seed()	настраивает генератор случайных чисел на новую последовательность. По умолчанию используется системное время. Если указать в скобочках необязательное значение параметра, то генерируется одна и та же последовательность
random.uniform(Начало, Конец)	возвращает псевдослучайное вещественное число в диапазоне от Начало до Конец
random.randint(Начало, Конец)	возвращает псевдослучайное целое число в диапазоне от Начало до Конец
random.choince(Последовательность)	возвращает случайный элемент из любой последовательности (строки, списка, кортежа)
random.randrange(Начало, Конец, Шаг)	возвращает случайно выбранное число из последовательности
random.shuffle(Список)	перемешивает последовательность (изменяется сама последовательность). Поэтому функция не работает для неизменяемых объектов

Вероятностные распределения. (Для умных взрослых, которые занимаются математикой и решили почитать наш справочник)

`random.triangular(low, high, mode)` — случайное число с плавающей точкой, `low ≤ N ≤ high`. `Mode` - распределение.

`random.betavariate(alpha, beta)` — бета-распределение. `alpha>0`, `beta>0`. Возвращает от 0 до 1.

`random.expovariate(lambd)` — экспоненциальное распределение. `lambd` равен 1/среднее желаемое. `Lambd` должен быть отличным от нуля. Возвращаемые значения от 0 до плюс бесконечности, если `lambd` положительно, и от минус бесконечности до 0, если lambd отрицательный.

`random.gammavariate(alpha, beta)` — гамма-распределение. Условия на параметры `alpha>0` и `beta>0`.

random.gauss(значение, стандартное отклонение) — распределение Гаусса.

`random.lognormvariate(mu, sigma)` — логарифм нормального распределения. Если взять натуральный логарифм этого распределения, то вы получите нормальное распределение со средним `mu` и стандартным отклонением `sigma`. `mu` может иметь любое значение, и `sigma` должна быть больше нуля.
`random.normalvariate(mu, sigma)` — нормальное распределение. `mu` — среднее значение, `sigma` — стандартное отклонение.
`random.vonmisesvariate(mu, kappa)` — `mu` — средний угол, выраженный в радианах от 0 до 2π, и `kappa` — параметр концентрации, который должен быть больше или равен нулю. Если каппа равна нулю, это распределение сводится к случайному углу в диапазоне от 0 до 2π.
`random.paretovariate(alpha)` — распределение Парето.
`random.weibullvariate(alpha, beta)` — распределение Вейбулла.

Примеры

Генерация произвольного пароля
Хороший пароль должен быть произвольным и состоять минимум из 6 символов, в нём должны быть цифры, строчные и прописные буквы. Приготовить такой пароль можно по следующему рецепту:

```
import random
# Набор цифр
str1 = '123456789'
# Набор строчных букв
str2 = 'qwertyuiopasdfghjklzxcvbnm'
# То же, но в верхнем регистре
str3 = str2.upper()
print(str3)
# Выведет: 'QWERTYUIOPASDFGHJKLZXCVBNM'
# Соединяем все строки в одну
str4 = str1+str2+str3
print(str4)
# Выведет:
'123456789qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM'
# Преобразуем получившуюся строку в список
ls = list(str4)
# Тщательно перемешиваем список
random.shuffle(ls)
# Извлекаем из списка 12 произвольных значений
psw = ''.join([random.choice(ls) for x in range(12)])
# Пароль готов
print(psw)
# Выведет: '1t9G4YPsQ5L7'
```

Этот же скрипт можно записать всего в две строки:

```
import random
print(''.join([random.choice(list('123456789qwertyuiopasdfghjklz
xcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) for x in range(12)]))
```

Данная команда является краткой записью цикла for, вместо неё можно было написать так:

```
import random
```

```
psw = '' # предварительно создаем переменную psw
for x in range(12):
    psw = psw + random.choice(list('123456789qwertyuiopasdfgh
    jklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM'))
print(psw)
# Выведет: Ci7nU6343YGZ
```

Данный цикл повторяется 12 раз и на каждом круге добавляет к строке psw произвольно выбранный элемент из списка.

Подробнее здесь: http://ps.readthedocs.io/ru/latest/random.html

а это «на потом»: проверка наличия элемента в словаре

```
my_dict = {'key': 'value'}
key_exists = my_dict.has_key('key') # Устаревший способ.
key_exists = 'key' in my_dict # Актуальный способ.
```

Черепашья графика в языке программирования Python

Для работы с черепашьей графикой необходимо подключить специальный модуль и создать черепаху на черепашьем поле:

```
import turtle
t=turtle.Pen()
```

Таблица функций и методов работы с черепашьей графикой. В таблице используем черепаху t.

Метод или команда	Что делает	Пример
up()	Поднятие "пера", чтобы не оставалось следа его при перемещении	t.up()
down()	Опускание "пера", чтобы при перемещении оставался след (рисовались линии)	t.down()
goto(x,y)	Перемещение "пера" в точку с координатами x,y в системе координат окна рисования	t.goto(50,20)
color ('цвет')	Установка цвета "пера" в значение, определяемое строкой цвета	t.color('blue') t.color('#0000ff')
bgcolor ('цвет')	Установка цвета фона в значение, определяемое строкой цвета. Вызываем не для Pen, а для turtle	turtle.bgcolor('#909090')
width(n)	Установка толщины "пера" в точках экрана	t.width(3)
forward(n)	Передвижение "вперёд" (в направлении острия стрелки) на n точек	t.forward(100)
backward(n)	Передвижение "назад" на n точек	t.backward(100)
right(k)	Поворот направо (по часовой стрелке) на k единиц	t.right(75)
left(k)	Поворот налево (против часовой стрелки) на k единиц	t.left(45)
radians()	Установка единиц измерения углов в радианы	t.radians()
degrees()	Установка единиц измерения углов в градусы (включён по умолчанию)	t.degrees()
circle(r)	Рисование окружности радиусом r точек из текущей позиции "пера". Если r положительно, окружность рисуется против часовой стрелки, если отрицательно — по часовой стрелке.	t.circle(40) t.circle(-50)
circle(r,k)	Рисование дуги радиусом r точек и	t.circle(40,45)

Метод или команда	Что делает	Пример
	углом k единиц.	t.circle(-50,275)
write ('строка')	Вывод текста в текущей позиции пера	t.write('Начало координат!')
clear()	Очистка области рисования	t.clear(0)

Прочитать подробнее можно здесь: https://www.intuit.ru/studies/courses/3489/731/lecture/25771

Файлы в языке программирования Python

Для работы с файлами будем использовать файловые переменные, которые будут представлять файлы в нашей программе.

```
ou = open('c://spam.txt', 'w')  - Открывает файл spam.txt для записи ('w' означает write – запись)

inp = open('data', 'r') -  Открывает файл data из той-же папки, в которой находится наша программа для чтения  ('r' означает read – чтение)
```

диалог открытия файлов
from PyQt5.QtWidgets import QFileDialog
name= QFileDialog.getOpenFileName(self, “text”,””,”pictions (*.jpg)”)[0]

Таблица функций и методов работы с файлами. В таблице используем описанные выше переменные ou и inp.

Пример команды	Что делает
a = inp.read()	Чтение файла целиком в строку a
a = inp.read(N)	Чтение следующих N символов (или байтов) в строку a
a = inp.readline()	Чтение следующей текстовой строки (включая символ конца строки) в строку a. Для удаления символа ‘\n’ из конца файла удобно использовать метод строки rstrip(). Например: a = a.rstrip()
Li = inp.readlines()	Чтение файла целиком в список строк Li (включая символ конца строки)
ou.write(S)	Запись строки символов S (или байтов) в файл
ou.writelines(Li)	Запись всех строк из списка Li в файл
ou.close()	Закрытие файла (выполняется по окончании работы с файлом)
ou.flush()	Выталкивает выходные буферы на диск, файл остается открытым
anyFile.seek(N)	Изменяет текущую позицию в файле для следующей операции, смещая ее на N байтов от начала файла.
for line in open(‘data’): операции над line	Переменная line в цикле по очереди принимает значение каждой строки файла data
f1=open(‘f.txt’, encoding=’latin-1’)	Открытие файла с кодировкой Юникод
f1=open(‘f.bin’, ‘rb’)	Открытие двоичного файла

Сохранение и интерпретация объектов Python в файлах

Следующий пример записывает различные объекты в текстовый файл.

```
>>> X, Y, Z = 43, 44, 45          # Объекты языка Python должны
>>> S = 'Spam'                    # записываться в файл только
                                    в виде строк
>>> D = {'a': 1, 'b': 2}
>>> L = [1, 2, 3]
>>>
>>> F = open('datafile.txt', 'w') # Создает файл для записи
>>> F.write(S + '\n')             # Строки завершаются символом
\n
>>> F.write('%s,%s,%s\n' % (X, Y, Z))# Преобразует числа в
строки
>>> F.write(str(L) + '$' + str(D) + '\n') # Преобразует
                                          и разделяет символом $
>>> F.close()
```

Использование инструкции `print`:

```
>>> chars = open('datafile.txt').read() # Отображение строки
>>> chars                          # в неформатированном виде
"Spam\n43,44,45\n[1, 2, 3]${'a': 1, 'b': 2}\n"
>>> print(chars)                    # Удобочитаемое
представление
Spam
43,44,45
[1, 2, 3]${'a': 1, 'b': 2}
```

Теперь нам необходимо выполнить обратные преобразования, чтобы получить из строк в текстовом файле действительные объекты языка Python.

```
>>> F = open('datafile.txt') # Открыть файл снова
>>> line = F.readline()      # Прочитать одну строку
>>> line
'Spam\n'
>>> line.rstrip()            # Удалить символ конца строки
'Spam'
```

Пока что мы прочитали ту часть файла, которая содержит строку. Теперь прочитаем следующий блок, в котором содержатся числа, и выполним разбор этого блока (то есть извлечем объекты):

```
>>> line = F.readline()        # Следующая строка из файла
>>> line                       # Это - строка
'43,44,45\n'
>>> parts = line.split(',')    # Разбить на подстроки по запятым
>>> parts
['43', '44', '45\n']
```

Здесь был использован метод split, чтобы разбить строку на части по запятым, которые играют роль символов-разделителей, – в результате мы получили список строк, каждая из которых содержит отдельное число. Теперь нам необходимо преобразовать эти строки в целые числа, чтобы можно было выполнять математические операции над ними:

```
>>> int(parts[1])       # Преобразовать строку в целое число
44
>>> numbers = [int(P) for P in parts] # Преобразовать весь
список
>>> numbers
[43, 44, 45]
```

Как мы уже знаем, функция int преобразует строку цифр в объект целого числа.

Наконец, чтобы преобразовать список и словарь в третьей строке файла, можно воспользоваться встроенной функцией eval, которая интерпретирует строку как программный код на языке Python:

```
>>> line = F.readline()
>>> line
"[1, 2, 3]${'a': 1, 'b': 2}\n"
>>> parts = line.split('$')  # Разбить на строки по символу $
>>> parts
['[1, 2, 3]', "{'a': 1, 'b': 2}\n"]
>>> eval(parts[0])           # Преобразовать строку в объект
[1, 2, 3]
>>> objects = [eval(P) for P in parts] # То же самое для всех
строк в списке
>>> objects
[[1, 2, 3], {'a': 1, 'b': 2}]
```

Поскольку теперь все данные представляют собой список обычных объектов, а не строк, мы сможем применять к ним операции списков и словарей.

Перегрузка специальных операторов в языке программирования Python

Все методы в Pithon первым входным параметром имеют self. Кроме того, обращение к полям класса тоже выполняется через self.

В python имеются методы, которые, как правило, не вызываются напрямую, а вызываются встроенными функциями или операторами. Например, __str__(self) - вызывается функциями str, print и format.

__**new**__(cls[, ...]) — управляет созданием экземпляра. В качестве обязательного аргумента принимает класс (не путать с экземпляром). Должен возвращать экземпляр класса для его последующей его передачи методу __init__.

__**init**__(self[, ...]) – конструктор класса.

__**del**__(self) - вызывается при удалении объекта сборщиком мусора.

__**repr**__(self) - вызывается встроенной функцией repr; возвращает "сырые" данные, использующиеся для внутреннего представления в python.

__**str**__(self) - вызывается функциями str, print и format. Возвращает строковое представление объекта.

__**bytes**__(self) - вызывается функцией bytes при преобразовании к байтам.

__**format**__(self, format_spec) - используется функцией format (а также методом format у строк).

__**lt**__(self, other) - x < other Если объект, для которого вызываем, меньше чем other, возвращать True, иначе – False и т.д.

__**le**__(self, other) - x ≤ y вызывает x.__le__(y).

__**eq**__(self, other) - x == y вызывает x.__eq__(y).

__**ne**__(self, other) - x != y вызывает x.__ne__(y)

__**gt**__(self, other) - x > y вызывает x.__gt__(y).

__**ge**__(self, other) - x ≥ y вызывает x.__ge__(y).

__**hash**__(self) - получение хэш-суммы объекта, например, для добавления в словарь.

__**bool**__(self) - вызывается при проверке истинности. Если этот метод не определён, вызывается метод __len__ (объекты, имеющие ненулевую длину, считаются истинными).

__**getattr**__(self, name) - вызывается, когда атрибут экземпляра класса не найден в обычных местах (например, у экземпляра нет метода с таким названием).

__**setattr**__(self, name, value) - назначение атрибута.

__**delattr**__(self, name) - удаление атрибута (del obj.name).

__**call**__(self[, args...]) - вызов экземпляра класса как функции.

__**len**__(self) - длина объекта.

__**getitem**__(self, key) - доступ по индексу (или ключу).

__**setitem**__(self, key, value) - назначение элемента по индексу.

__**delitem**__(self, key) - удаление элемента по индексу.

__**iter**__(self) - возвращает итератор для контейнера.

__**reversed**__(self) - итератор из элементов, следующих в обратном порядке.

__**contains**__(self, item) - проверка на принадлежность элемента контейнеру (item in self).

Перегрузка арифметических операторов

__**add**__(self, other) - сложение. x + y вызывает x.__add__(y).

__**sub**__(self, other) - вычитание (x - y).

__**mul**__(self, other) - умножение (x * y).

__**truediv**__(self, other) - деление (x / y).

__**floordiv**__(self, other) - целочисленное деление (x // y).

__**mod**__(self, other) - остаток от деления (x % y).

__**divmod**__(self, other) - частное и остаток (divmod(x, y)).

__**pow**__(self, other[, modulo]) - возведение в степень (x ** y, pow(x, y[, modulo])).

__**lshift**__(self, other) - битовый сдвиг влево (x << y).

__**rshift**__(self, other) - битовый сдвиг вправо (x >> y).

__**and**__(self, other) - битовое И (x & y).

__**xor**__(self, other) - битовое ИСКЛЮЧАЮЩЕЕ ИЛИ (x ^ y).

__**or**__(self, other) - битовое ИЛИ (x | y).

Пойдём дальше.

__**radd**__(self, other),

__**rsub**__(self, other),

__**rmul**__(self, other),

__**rtruediv**__(self, other),

__**rfloordiv**__(self, other),

__**rmod**__(self, other),

__**rdivmod**__(self, other),

__**rpow**__(self, other),

__**rlshift**__(self, other),

__**rrshift**__(self, other),

__**rand**__(self, other),

__**rxor**__(self, other),

__**ror**__(self, other) - делают то же самое, что и арифметические операторы, перечисленные выше, но для аргументов, находящихся справа, и только в случае, если для левого операнда не определён соответствующий метод.

Например, операция x + y будет сначала пытаться вызвать x.__add__(y), и только в том случае, если это не получилось, будет пытаться вызвать y.__radd__(x). Аналогично для остальных методов.

При реализации этих методов нужно возвращать новый объект.

Идём дальше.

__**iadd**__(self, other) - +=.

__**isub**__(self, other) - -=.

__**imul**__(self, other) - *=.

__**itruediv**__(self, other) - /=.

__**ifloordiv**__(self, other) - //=.

__**imod**__(self, other) - %=.

__**ipow**__(self, other[, modulo]) - **=.

__**ilshift**__(self, other) - <<=.

__**irshift**__(self, other) - >>=.

__**iand**__(self, other) - &=.

__**ixor**__(self, other) - ^=.

__**ior**__(self, other) - |=.

__**neg**__(self) - унарный -.

__**pos**__(self) - унарный +.

При реализации этих методов следует возвращать self.

Идём дальше.

__**abs**__(self) - модуль (abs()).

__**invert**__(self) - инверсия (~).

__**complex**__(self) - приведение к complex.

__**int**__(self) - приведение к int.

__**float**__(self) - приведение к float.

__**round**__(self[, n]) - округление.

__**enter**__(self), __**exit**__(self, exc_type, exc_value, traceback) - реализация менеджеров контекста.

Рассмотрим некоторые из этих методов на примере двухмерного вектора, для которого переопределим некоторые методы:

```
import math

class Vector2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return 'Vector2D({}, {})'.format(self.x, self.y)

    def __str__(self):
        return '({}, {})'.format(self.x, self.y)

    def __add__(self, other):
        return Vector2D(self.x + other.x, self.y + other.y)

    def __iadd__(self, other):
        self.x += other.x
        self.y += other.y
        return self

    def __sub__(self, other):
        return Vector2D(self.x - other.x, self.y - other.y)

    def __isub__(self, other):
        self.x -= other.x
        self.y -= other.y
        return self

    def __abs__(self):
        return math.hypot(self.x, self.y)

    def __bool__(self):
        return self.x != 0 or self.y != 0

    def __neg__(self):
        return Vector2D(-self.x, -self.y)

>>> x = Vector2D(3, 4)
>>> x
Vector2D(3, 4)
```

```
>>> print(x)
(3, 4)
>>> abs(x)
5.0
>>> y = Vector2D(5, 6)
>>> y
Vector2D(5, 6)
>>> x + y
Vector2D(8, 10)
>>> x - y
Vector2D(-2, -2)
>>> -x
Vector2D(-3, -4)
>>> x += y
>>> x
Vector2D(8, 10)
>>> bool(x)
True
>>> z = Vector2D(0, 0)
>>> bool(z)
False
>>> -z
Vector2D(0, 0)
```

Создание пользовательского интерфейса с помощью PyQt5

консоль от имени администратора в любом случае

`pip install PyQt5`

`pip install pyqt5-tools` после установки данного пакета, из пуска можно запускать designer.exe

Можно из дизайнера получить код:

запускаем с шифтом правой кнопкой мыши из папки с проектом командное окно и далее `pyuic5 -d -x -o demo.py demo.ui`

здесь -x это добавление кода до рабочего проекта. -o filedest это имя результирующего файла. Имя файла д.б. латиницей.

pyuic5 -h это помощь

Подключение:

1) встраивание

В свой код вставляем полученный класс.

Добавляем

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QFileDialog
```

создаем свой класс, наследуясь от сгенерированного и от QMainWindow

```
class Form1(Ui_MainWindow, QMainWindow):
    def __init__(self):
        super().__init__()
        self.setupUi(self)


if __name__ == '__main__':
    app = QApplication(sys.argv)
    form1 = Form1()
    form1.show()
    sys.exit(app.exec_())
```