ЯЗЫК ПРОГРАММИРОВАНИЯ PYTHON

Контейнерные классы, исключения и файлы в языке Python

Крашенинников Роман Сергеевич
Главный специалист отдела системного администрирования РХТУ им. Д.И. Менделеева, ведущий программист кафедры информационных компьютерных технологий

ТЕМЫ

- Списки
- Кортежи
- Строки
- Словари
- Работа с файлами
- Исключения и способы их обработки

ПОЛЕЗНЫЕ РЕСУРСЫ

- Коллекции официальная документация
- Исключения документация
- Форматирование строк
- Строки
- Списки
- Словари
- Файлы
- Формат JSON

СПИСКИ

Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы могут отличаться).

Генерировать списки можно различными способами. Можно использовать встроенную функцию **list**, литерал [] или же генератора списков.

Для обращения к элементу списка, номер элемента передается в []. Важно помнить, что в **Python** нумерация начинается с 0!

Создание при помощи **list**

```
l = list('список')
# ['с', 'п', 'и', 'с', 'о', 'к']
```

Создание при помощи []

```
l = []
s = [1, 'h' , False]
```

Создание при помощи генератора

```
l = [c for c in 'список']
# ['с', 'п', 'и', 'с', 'о', 'к']
```

Обращение к элементу

```
s = [1, 'h' , False]
print(s[1])
```

СПИСКИ (МЕТОДЫ)

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

СПИСКИ (СРЕЗЫ)

Вывод второго элемента с конца

```
a = [1, 2, 3, 4, 5]
print(a[-2])
```

Вывод элементов с третьего по шестой

```
a = [1, 2, 3, 4, 5, 6, 7]
print(a[2:5])
```

Вывод элементов на четных позициях

```
a = [1, 2, 3, 4, 5]
print(a[::2])
```


```
a = [1, 2, 3, 4, 5]
print(a[::-1])
```

Вывод элементов на четных позициях со второго (не включая) по шестой в обратном порядке

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(a[7:1:-2])
```

СПИСКИ (РАСПАКОВКА)

Списки можно распаковывать, это позволяет помещать элементы списка в отдельные переменные. Стоит помнить, что при полной распаковке, количество переменных должно равняться количеству элементов в списке. Кроме того, распаковка позволяет объединять списки.

```
#Полная распаковка
b, c, d, e = a
print(b, c, d, e)

#Частичня распаковка
*b, c, d = a
print(b, c, d)
```

```
1 2 3 4
[1, 2] 3 4
```

Объединение списков

```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
b = [4, 11, 17, 52]
print([*a, *b])
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 4, 11, 17, 52]
```

КОРТЕЖИ

Кортеж – по сути является неизменяемым списком. Для чего нужен? Защищает от случайных или намеренных изменений а так же экономит память. Плюс к этому, кортежи можно использовать как ключи словарей.

Объявить кортеж можно при помощи функции **tuple** или же ().

Кортежи поддерживают срезы и распаковку, но не поддерживают функции списков.


```
c = (5, 6, 7)
c_tuple = tuple("кортеж")

print(c, c_tuple)
```

СТРОКИ

Строки в Python - упорядоченные последовательности символов, используемые для хранения и представления текстовой информации, поэтому с помощью строк можно работать со всем, что может быть представлено в текстовой форме.

Объявить строку можно при помощи кавычек или же функции **str** (если нужно перевести другой тип данных в строку)

Создание при помощи “”

```
s = "Hello world !"
print(s)
```

Создание при помощи **str**

```
s = str(737373)
print(s)
```

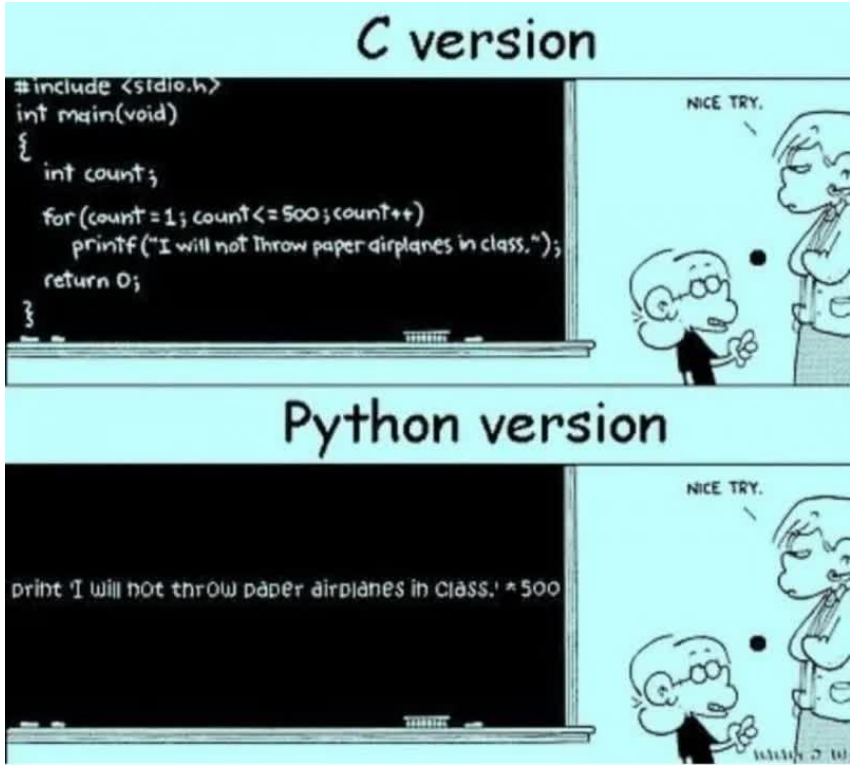
Создание при помощи ‘’

```
s = 'Hello world !'
print(s)
```

СТРОКИ (ОСОБЕННОСТИ)

Сравнение строк

```
1 a = "Some text"
2 b = 'Some text'
3 print(a, b)
4 print(a == b)
```



Кавычки внутри кавычек

```
1 a = "Can't use \"."
2 b = 'Can\'t use \'.'
3 print(a)
4 print(b)
```

Сложение строк

```
s1 = "Hello"
s2 = "World"
s3 = s1 + s2
print(s3)
```

Умножение строки на число

```
s1 = "A"
s2 = 25 * s1
print(s2)
```

СТРОКИ (МЕТОДЫ)

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(шаблон, замена[, maxcount])	Замена шаблона на замену. maxcount ограничивает количество замен
S.split(символ)	Разбиение строки по разделителю
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'))

СТРОКИ (МЕТОДЫ)

S.istitle()	Начинаются ли слова в строке с заглавной буквы
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру
S.startswith(str)	Начинается ли строка S с шаблона str
S.endswith(str)	Заканчивается ли строка S шаблоном str
S.join(список)	Сборка строки из списка с разделителем S
ord(символ)	Символ в его код ASCII
chr(число)	Код ASCII в символ
S.capitalize()	Переводит первый символ строки в верхний регистр, а все остальные в нижний
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

СТРОКИ (МЕТОДЫ)

S.lstrip([chars])	Удаление пробельных символов в начале строки
S.rstrip([chars])	Удаление пробельных символов в конце строки
S.strip([chars])	Удаление 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()	Переводит символы нижнего регистра в верхний, а верхнего – в нижний
S.title()	Первую букву каждого слова переводит в верхний регистр, а все остальные в нижний
S.zfill(width)	Делает длину строки не меньшей width, по необходимости заполняя первые символы нулями
S.ljust(width, fillchar=" ")	Делает длину строки не меньшей width, по необходимости заполняя последние символы символом fillchar
S.rjust(width, fillchar=" ")	Делает длину строки не меньшей width, по необходимости заполняя первые символы символом fillchar
S.format(*args, **kwargs)	Форматирование строки

СЛОВАРИ

Словари в Python - неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами. Объявить словарь можно с помощью литерала {} или функции **dict.**

Создание при помощи **dict**

```
d = dict(a=1, b=2)
print(d)
```

Создание при **dict (x2)**

```
d = dict([('a', 1), ('b', 2)])
print(d)
```

Создание при помощи {}

```
d = {'a': 1, 'b': 2}
print(d)
```

СЛОВАРИ (МЕТОДЫ)

dict.clear() - очищает словарь.

dict.copy() - возвращает копию словаря.

classmethod **dict.fromkeys**(seq[, value]) - создает словарь с ключами из seq и значением value (по умолчанию None).

dict.get(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).

dict.items() - возвращает пары (ключ, значение).

dict.keys() - возвращает ключи в словаре.

dict.pop(key[, default]) - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).

dict.popitem() - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.

dict.setdefault(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).

dict.update([other]) - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).

dict.values() - возвращает значения в словаре.

СЛОВАРИ (РАСПАКОВКА)

Распаковка может быть применена и для словарей. С помощью распаковки можно легко объединить несколько списков. В случае наличия одинаковых элементов, присваиваются значения наиболее позднего вхождения. Иногда с использованием распаковки удобно передавать параметры в функцию (смотри в следующей серии…).

Объединение словарей распаковкой

```
1 a = {"a": 1, "b": 2}
2 b = {"b": 3, "c": 4}
3 c = {**a, **b}
4 print(c)
```

СЛОВАРИ (ПЕРЕЧИСЛЕНИЕ)

По умолчанию перечисление словарей осуществляется только по ключам. Однако, с помощью автоматической распаковки на каждой итерации цикла, можно осуществлять одновременное перечисления по ключам и значениям словаря

Без распаковки

```
1 a = {"a": 1, 2: "b", 3.5: [1, 2]}
2 # Только по ключам
3 for el in a:
4     print(el)
5 # С обращением по ключам
6 for el in a:
7     print(el, a[el])
```

С распаковкой

```
1 a = {"a": 1, 2: "b", 3.5: [1, 2]}
2 for k, v in a.items():
3     print(k, v)
```

ФАЙЛЫ

Файлы помогают программисту упростить процедуру ввода и сохранения различной информации, необходимой для работоспособности приложения.

Для открытия файлов используется встроенная функция open, в неё передается путь к файлу и режим открытия.

```
f = open("file.txt", "r")
```

Путь к файл Режим открытия

Режим	Обозначение
'r'	открытие на чтение (является значением по умолчанию).
'w'	открытие на запись, содержимое файла удаляется, если файла не существует, создается новый.
'x'	открытие на запись, если файла не существует, иначе исключение.
'a'	открытие на дозапись, информация добавляется в конец файла.
'b'	открытие в двоичном режиме.
't'	открытие в текстовом режиме (является значением по умолчанию).
'+'	открытие на чтение и запись

ФАЙЛЫ(ЧТЕНИЕ И ЗАПИСЬ)

Чтение из файла осуществляется при помощи функции read, в неё можно передать количество символов, которое нужно считать, если не передавать ничего, считается всё из файла. Для записи в файл применяется функция write. После работы с файлом рекомендуется закрыть функцией close.

Запись данных в файл

```
>>> f = open("file.txt", 'w')
>>> stroka = "qwerty"
>>> f.write(stroka)
>>> f.close()
```

Чтение данных из файла

```
>>> f = open("file.txt", 'r')
>>> print(f.read())
qwerty
>>> f.close()
```

ФАЙЛЫ(ЗАКРЫТИЕ)

Для рационального использования ресурсов, рекомендуется всегда закрывать файлы, предпочтительно и наиболее просто это делается при помощи конструкции with.

Использование **with**

```
with open("file.txt", "r") as f:
    print(f.read())
```

ФАЙЛЫ(ФОРМАТ JSON)

JSON (JavaScript Object Notation) - простой формат обмена данными, основанный на подмножестве синтаксиса JavaScript. Модуль json позволяет кодировать и декодировать данные в удобном формате. По сути формат JSON – это хранение данных в виде словарей. Модуль широкий функционал, но для базового использования можно обойтись функциями load (чтение данных) и dumps (запись данных).

Запись данных

```
import json


my_data = {
    "a": 15,
    "b": "hello",
    "c": [1, 2, 3]
    }

with open("file.json", "w") as f:
    f.write(json.dumps(my_data))
```

Чтение данных

```
import json


with open("file.json", "r") as f:
    my_data = json.load(f)
    print(my_data)
```

ИСКЛЮЧЕНИЯ

Исключения - это события возникающие в случаях когда интерпретатор языка Python встречается с проблемой которую он не в состоянии решить самостоятельно. Знание основ работы с исключениями в Python является важным навыком т.к. ни одна программа не застрахована от ошибок. Полный список исключений можно найти тут.

```
print(1 / 0)
```

```
Traceback (most recent call last):

  File "C:\Users\Roman\.spyder-py3\temp.py", line 2, in <module>
    print(1 / 0)

ZeroDivisionError: division by zero
```

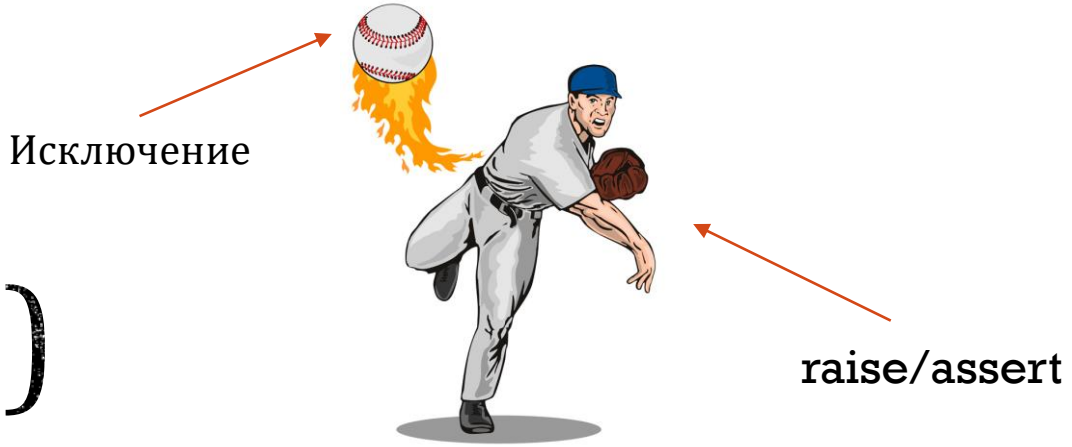
```
print(5 + '4')
```

```
Traceback (most recent call last):

  File "C:\Users\Roman\.spyder-py3\temp.py", line 3, in <module>
    print(5 + '4')

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

ИСКЛЮЧЕНИЯ (ВЫБРАСЫВАНИЕ)



Существует ряд случаев когда возникновение ошибки является желаемым результатом. Например, при проверке начальных условий какого-либо алгоритма, желательным исходом в случае их несоответствия является возникновение ошибки, а не “молчаливое завершение алгоритма”. Существует два ключевых слова позволяющих выбрасывать пользовательские исключение: raise и assert.

```
1 assert 1 == 1
2 assert 2 > 3
```

assert позволяет выбрасывать AssertionError если не выполняется условие указанное после оператора.

```
1 raise Exception("Some text")
```

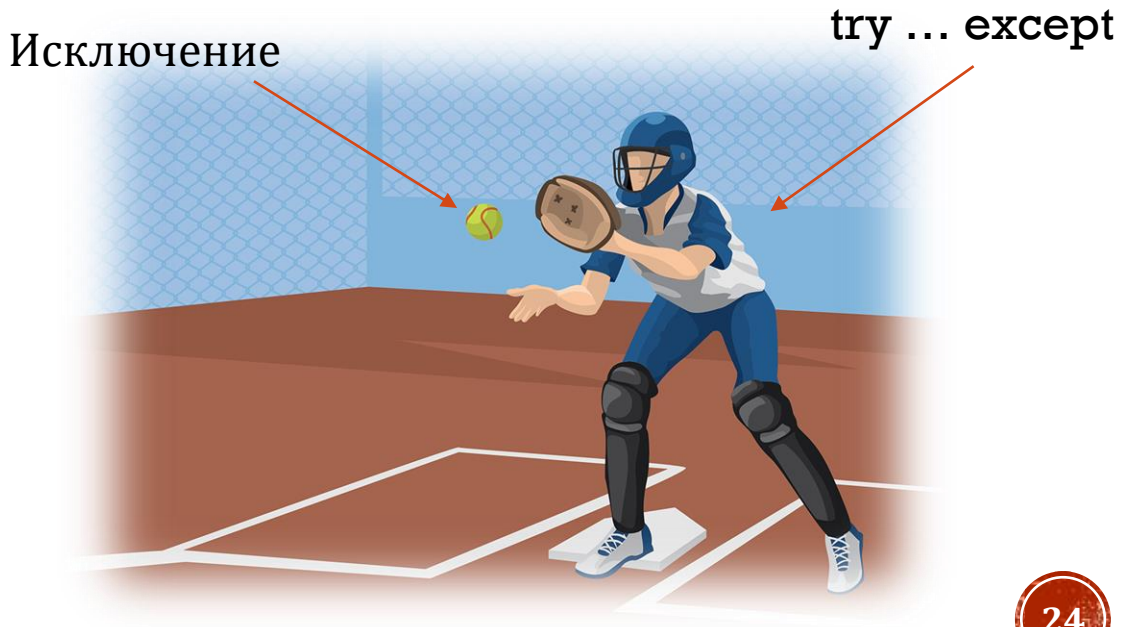
raise позволяет выбрасывать любое исключение, класс которого указывается после оператора

ИСКЛЮЧЕНИЯ (ОБРАБОТКА)

Программа, которая прекращает свое выполнение после первой же ошибки без сообщения конечному пользователю о причинах ее возникновения, является плохой программой Для отлавливания исключений и попытки их программного исправления или формирования отчета существует конструкция try ... except.

Общий вид конструкции

```
try:
    <тело оператора>
except <класс ошибки> as <переменная>:
    <код обработки исключения>
except <класс ошибки>:
    <код обработки исключения>
except:
    <код обработки исключения>
else:
    <когда нет исключения>
finally:
    <выполняется всегда>
```



ЗАДАНИЕ ДЛЯ САМОКОНТРОЛЯ

- Задать список a равный [-3, -19, 4, -15, 2, -10, -3, 1, -11, 19]
- Вывести каждый третий элемент.
- Сделать срез списка от **7** элемента до начала списка.
- Развернуть список встроенной функцией.
- Отсортировать список.
- Задать словарь d равный {'t': -18, 'b': -5, 'h': -18, 'y': 16, 'a': 6}
- Вывести все ключи словаря
- Создать словарь имеющий два новых ключа и один старый (любые ключи, любые значения). Обновить ваш словарь, с помощью нового, используя распаковку.
- Вывести список пар ключ значение.
- Добавить в словарь **d** список под ключом **a** и вывести словарь в файл **json**

СПАСИБО ЗА ВНИМАНИЕ!