Модуль Collections

OrderedDict

Гарантирует сохранение ключей в словаре в порядке добавления

```
from collections import OrderedDict
# Создание OrderedDict из списка
# кортежей формата (ключ, значение):
od = OrderedDict([('Ivan', 23),
   ('Nikita', 24), ('Elena',29),
   ('John',27), ('Gosha','22')])
# С версии Python 3.7 сохранение порядка
# ключей гарантируется и для обычных
# словарей
```

Deque

Возможности стека и очереди в одной структуре данных

```
from collections import deque
# Создание дэка из списка с макс. длиной
d = deque([1,2,3], maxlen=5)
# Добавление элемента справа и слева:
d.append(10); d.appendleft(4)
# Удаление элемента справа и слева:
print(d.pop(), d.popleft())
# 10 4
# Добавление нескольких элементов сразу:
d.extend([4,5,6]); d.extendleft([8,9])
# Развернуть очередь:
d.reverse()
# Переместить n элементов в начало:
d.rotate(3)
# Из начала в конец:
d.rotate(-3)
```

Defaultdict

Словарь, в котором по новым ключам генерируются объекты по умолчанию

```
from collections import defaultdict
# Создание пустого defaultdict со
# структурой list по умолчанию:
d = defaultdict(list)
# Можно сразу добавить значение в список
# по новому ключу:
d['new_key'].append(123)
# Обращение к несуществующему ключу
# приводит к добавлению ключа:
print(d['another_key'])
# []
print(d)
# defaultdict(<class 'list'>, {'new_key':
# [123], 'another_key': []})
```

Counter

Счётчик элементов в итерируемом объекте

```
from collections import Counter
# Создать счётчики из списка:
a = Counter([1,2,3,3,2,2])
b = Counter([1,1,4,3,2,3])
# Сложить счётчики:
print(a + b)
# Counter({2: 4, 3: 4, 1: 3, 4: 1})
# "-" не сохраняет разницу < 1:
print(a - b)
# Counter({2: 2})
# Вычесть из одного счётчика другой
# с учётом отрицательных чисел:
a.subtract(b)
# n наиболее частых значений:
b.most_common(3)
```

Модуль NumPy

Типы данных в NumPy

```
# 1 байт – 8 бит
# Целочисленные:
int8, int16, int32, int64
# Целочисленные неотрицательные:
uint8, uint16, uint32, uint64
# С плавающей точкой:
float16, float32, float64, float128
# Справка о типе данных:
np.iinfo(<целый тип данных>)
np.finfo(<тип с плавающей точкой>)
# Строковые и булевые NumPy-типы:
np.str_, np.bool_
```

Векторы

```
a = [3, 4, 2]
b = [12, 5, 8]
# Векторы из диапазона чисел:
np.range(<от>, <до>, <шаг>, dtype=<тип>)
np.linspace(<от>, <до>, <число>, dtype)
# Арифметика:
a + b; a - b; a * b; a * 3; ...
# Линейная алгебра:
np.linalg.norm(a) # Длина
np.linalg.norm(a-b) # Расстояние
np.dot(a, b) # Скалярное произведение
# Сортировка:
np.sort(a) # Возвращает новый вектор
a.sort() # Изменяется исходный вектор
# Статистика:
np.min(a) # Минимальное значение
np.max(a) # Максимальное значение
np.mean(a) # Среднее значение
np.sum(a) # Сумма значений
```

Массивы

```
# Массив из чисел без и с типом данных:
arr = np.array([14, 23, 56])
arr = np.array([14, 23], dtype=np.int8)
# Массив из нулей заданной формы:
arr = np.zeros(10)
arr = np.zeros((3, 4), dtype=np.int32)
# Изменение формы массива:
arr.shape = (2,4) # Меняется форма arr
arr.reshape(<форма>) # Возвращается новый
arrT = arr.transpose() # Строки ↔ столбцы
# Размерность, форма, число эл-ов:
arr.ndim; arr.shape; arr.size
# Индексация и срезы в многомерном массиве:
nd_arr[1][4] = nd_arr[1, 4]
nd_arr[:, 2]   nd_arr[:3, 2:4]
# Заполнить пропущенные значения нулями:
arr[np.isnan(arr)] = 0
```

Случайные числа

```
# Все функции из подмодуля np.random
# Задать seed генерации случайных чисел:
seed(121223) # Аргумент – uint32
# Случайные float от 0 до 1:
rand() # 1 число
rand(10) # 10 чисел
rand(12, 5) # 12x5 чисел
# Эквивалентно:
sample((12, 5)) # Форма – кортеж
# Float из диапазона:
uniform(<от>, <до>, size=<форма>)
# Int из диапазона:
randint(<от>, <до>, size=<форма>)
# Выборки:
shuffle(arr) # Перемешать тот же массив
permutation(arr) # Получить перемешанный
# Выбрать часть объектов из массива:
choice(<массив или число>, size=<форма>,
   replace=<Возможен повтор элементов?>)
```