Язык Python. Часть 2

Лекция 10

На чем остановились

- Библиотека numpy
- Работа с числовыми данными (в первую очередь)
- С векторами, матрицами
- Более многомерными структурами

Пример многомерной структуры

- Изображения
- Каждое изображение - прямоугольник
- Прямоугольник состоит из пикселей
- Каждый пиксель - цвет (3 компонента)
- Получаем трехмерные данные

ndarray и shape

- ndarray - основная структура данных
- Экземпляры могут иметь разную размерность
- shape - кортеж, описывающий размер по каждому измерению
- Размер shape равен размерности ndarray

Простейший пример

```
1 import numpy as np
2
3 data = np.ones(shape=(3, 2))
4 print(data)
5 data = np.zeros(shape=(3, 2, 4))
6 print(data)
7 data = np.ndarray(shape=(5, )) # что было в памяти - то и ув
8 print(data)
```

Преобразование стандартных последовательностей в ndarray

- Передаем параметром в np.array(...)
- Важна "прямоугольность"
- В широком смысле слова
- То есть, в многомерном случае - тоже
- Иначе будет не то, что хотелось

Пример преобразования

```
1 import numpy as np
2
3 data = np.array([1, 2])
4 print(data)
5 data = np.array([[1, 2], [3, 4]])
6 print(data)
```

Пример с нарушенной прямоугольностью

```
1 import numpy as np
2
3 data = np.array([1, 2])
4 print(data)
5 data = np.array([[1, 2], [3, 4, 3]])
6 print(data)
7 print(data.shape)
```

Поймем, что случилось

- Входная коллекция двумерна
- Но не прямоугольна
- Мы создали одномерный ndarray
- В котором храним ссылки на Python-списки
- Мы даже до чисел можем добраться

Пример доступом к числам при нарушенной прямоугольности

```
1 import numpy as np
2
3 data = np.array([[1, 2], [3, 4, 3]])
4 print(data[0][0])
5 print(data[0][1])
6 print(data[1][0])
7 print(data[1][1])
8 print(data[1][2])
9 print(data.shape)
```

Атрибуты ndarray

- ndim - количество измерений (длина shape)
- shape - уже знаем
- size - общее количество элементов
- (произведение элементов shape)
- dtype - тип элемента
- itemsize - размер элемента (определяется типом)

Пример с ненарушенной прямоугольностью

```
1 import numpy as np
2
3 data = np.array([[1, 2, 3], [4, 5, 6]])
4 print(data.ndim)
5 print(data.shape)
6 print(data.size)
7 print(data.dtype)
8 print(data.itemsize)
```

Пример с нарушенной прямоугольностью

```
1 import numpy as np
2
3 data = np.array([[1, 2], [4, 4, 6]])
4 print(data.ndim)
5 print(data.shape)
6 print(data.size)
7 print(data.dtype)
8 print(data.itemsize)
```

Числовые и нечисловые ndarray

- Можем создать ndarray строк
- Или произвольных объектов
- Формально все так же
- Многомерность, shape, и т.п.
- Но есть существенное отличие от числовых

Числовые и нечисловые массивы

- Числовые реализованы очень быстро
- От Python - только интерфейс
- Реализация - на C
- С использованием возможностей процессора

Числовые и нечисловые массивы

- Для работы с числовыми данными есть массированные операции
- Работающие с векторами и матрицами в целом
- Надо всегда пользоваться ими
- Не пытаться писать свои циклы ради арифметики

Система типов

- Основные группы типов - целые и вещественные
- Есть типы с абсолютным размером и относительным
- Сфокусируемся на абсолютном размере
- Начнем с целых
- Самый маленький размер - 8 битов

Система типов

- Целые типы бывают знаковые и беззнаковые
- 8-битовые: int8(знаковый) и uint8(беззнаковый)
- Диапазон значений
- [0, 255] для беззнакового
- [-128, 127] для знакового

Выход за границы диапазона

- Примерно как обнуление спидометра
- Начинаем считать с 0
- Или с -128
- 255 + 1 даст 0
- 253 + 4 даст 2

Пример на uint8

```
1 import numpy as np
2
3 data = np.array([1, 2], dtype=np.uint8)
4 print(data)
5 data[0] += 255
6 print(data) # [0, 2]
```

Пример на int8

```
1 import numpy as np
2
3 data = np.array([1, 2], dtype=np.int8)
4 print(data)
5 data += 127
6 print(data) # [-128, -127]
```

Другие целые типы

- int16, uint16 - по аналогии с восьбитовыми
- Только диапазон чисел больше
- 2 в 16 степени
- Примерно 64К (64 * 1024)

Другие целые типы

- int32, uint32 - ничего нового
- Только диапазон чисел еще больше
- Аналогично с int64, uint64
- Есть особый случай со знаковыми типами

Тип - свойство ndarray

- По умолчанию ones/zeros создают ndarray типа float64
- array - в зависимости от типа значений
- Если там только целые - будет int64
- Иначе - float64

Тип - свойство ndarray

- Но можем указать тип явно
- Через параметр dtype
- Важно, чтобы значения влезали в этот тип
- Иначе получим переполнение

Пример на переполнение в момент создания

```
1 import numpy as np
2
3 data = np.array([100, 200, 300], dtype=np.int8)
4 print(data) # [1, -56, 44]
```

Скалярная вырезка - тоже ndarray

- Если взять элемент ndarray
- И сохранить в Python-переменной
- Это будет НЕ целое или вещественное в Python-смысле
- Это будет ndarray с пустым shape

Пример

```
 1 import numpy as np
 2
 3 data = np.array([100, 200, 20], dtype=np.int32)
 4 v1 = data[0]
 5 print(type(v1))
 6 print(v1.dtype)
 7 print(v1.shape)
 8 v2 = data[1]
 9 data2 = np.array([v1, v2])
10 print(data2.dtype) # np.int32
11 data2 = np.array([100, 200])
12 print(data2.dtype) # np.int64
```

Изменяемость ndarray

- Python-списки - изменяемы
- Кортежи, строки - неизменяемы
- ndarray - посередине
- Неизменяемы по структуре
- Изменяемы по содержимому

Массированные операции

- Правильный шаблон использования numpy - массированные операции
- Чем меньше явных циклов, тем лучше
- Простейшее - присваивание значения всему ndarray
- Инкремент/декремент

Пример

```
 1 import numpy as np
 2
 3 data = np.array([100, 200, 20], dtype=np.int16)
 4 print(data)
 5 print(data + 1)
 6 data += 2
 7 print(data)
 8 print(data)
 9 data[:] = 11
10 print(data)
11 data = 1.23 # больше не ndarray
12 print(data)
```