Типы и структуры данных

Типы, коллекции, условные операторы, циклы

Полезная литература

- Марк Лутц, Изучаем Python (Learning Python)
- Hitchhikers guide to Python (https://docs.python-guide.org/)
- Python Notes for professionals (https://books.goalkicker.com/PythonBook/)
- Micha Gorelick & Ian Ozsvald, High Performance Python

Оператор присваивания

a = 10

- Создается объект типа int (ему дается id), со значением 10
- Переменная a теперь ссылается на этот объект
- Если объект изменит свое значение – изменится и то значение, которое мы получаем обращаясь к переменной a

Все объект

```
a = 'asdf'

print(type(a))  # <class 'str'>
print(id(a))  # 439849109

class Name(str):
    def get_first_name(self):
        return self.split(' ')[0]


n = Name('Порфирий Петрович')

print(n.get_first_name())  # Порфирий
```

- У всего есть id (уникальное число в рамках запущенного процесса)
- «примитивные типы» мало чем отличаются от пользовательских классов

Изменяемые и неизменяемые типы

Immutable

- str
- int, float, complex
- bool
- tuple

Mutable

- list
- dict
- set
- пользовательские объекты

Оператор is

```
a = 4
b = a
a is b  # True

a = 3
b = 1 + 2
a is b  # True --but that is only true for values under 256

l1 = []
l2 = []
l1 is l2  # False
```

int

- Не ограничен в размере
- Операции с int могут не сохранить тип

```
print(type(5 / 2))  # <class 'float'>

print(5 // 2)  # 2

print(5 % 2)  # 1
```

bool

Является частным случаем int (унаследован от него)

```
True + False  # 1
True / False  # ZeroDivisionError: division by zero

True is 1  # False
True == 1  # True
```

Метод bool

```
bool(5)  # True
bool(0)  # False
bool('')  # False
bool([])  # False
```

Falsey value

- Если объект «имеет длину» (т.е. можно применить метод len) и она равна нулю – булево значение объекта - False

Операторы and и or

- Они «ленивые»
- Если первый аргумент уже определил значение, он будет возвращен, а второй даже не вычисляется (в коде ниже никаких делений на ноль выполнено не будет)

```
[1,2,3] or 5 / 0  # [1, 2, 3]
0 and 5 / 0  # 0


1 and 2  # 2
```

- Если по первому аргументу понять значение нельзя – вернется вычисленный второй аргумент

If, elif, else

```
# n = 6
if n % 2 == 0:
    print('n is even')
if n % 3 == 0:
    print('n is divisible by 3')
else:
    print('n is not divisible by 2 or 3')
# n is divisible by 2
# n is divisible by 3
```

```
# n = 6
if n % 2 == 0:
    print('n is even')
elif n % 3 == 0:
    print('n is divisible by 3')
else:
    print('n is not divisible by 2 or 3')
# n is divisible by 2
```

str

- Одинарные и двойные кавычки синтаксически идентичны
- Они позволяют нам не делать лишних экранирований

```
question = 'Have you read "Gone with the wind"?'
answer = "No, I haven't"
```

```
statement = '''
    I could be an sql statement
    or just a long string constant
    or docstring
'''
```

Обращение по индексу и конкатенация строк

Не забывайте, что строки не изменяемы!

```
answer = "No, I haven't"
answer[4]  # I
answer[4] = 'i'  # TypeError: 'str' object does not support item assignment

full_answer = answer + ' ' + 'read it yet'
print(answer)  # # No, I haven't
print(full_answer)  # No, I haven't read it yet
```

Еще есть много полезных методов

```
print(answer.lower())  # no i haven't
print(answer.upper())  # NO I HAVEN'T
print(answer.split(' '))  # ['No,', 'I', "haven't"]
print(answer.find('I'))  # 4
print(answer.replace('I', 'you'))  # No, you haven't

print('_'.join(['a', 'b', 'c']))  # a_b_c
print('        \ntable row\n       '.strip())  # table row
'005234501'.isdigit()  # True

'%' * 20  # %%%%%%%%%%%%%%%%%%%%
```

list

```
lst = []  # lst = list()

lst.append('element')
lst[0]  # 'element'
lst[1]  # IndexError...

lst.extend([5, 100500])  # ['element', 5, 100500]
lst.pop(1)  # 5
lst  # ['element', 100500]

lst + [200600]  # ['element', 100500, 200600]
lst  # ['element', 100500]
lst += [200600]
lst  # ['element', 100500, 200600]
```

list

```
lst = [[1], [2]]
long_lst = lst * 10  # [[1], [2], [1], [2], [1], [2], [1], [2]...]
lst[1][0] = 'foo'
long_lst  # [[1], ['foo'], [1], ['foo'], [1], ['foo'], [1], ['foo']...]

lst.reverse()  # [200600, 100500, 'element']
lst.sort()  # TypeError: '<' not supported between instances of 'str' and 'int'

lst.insert(2, 'blob')  # [200600, 100500, 'blob', 'element']
lst.index('element')  # 3
```

tuple

```
tpl = (1,)  # tpl = tuple([1])
tpl[0]  # 1
tpl[0] = 5  # TypeError: 'tuple' object does not support item assignment

tpl = ([1, 2],)
tpl[0][1] = 7
tpl  # ([1, 7],)
```

Также поддерживает slicing

set

- Гарантирует уникальность объектов

```
s = {'a', 'b', 'c', 'd'}  # s = set('abcd')
s.add(33)
s.add('a')
s  # {'a', 'b', 'c', 'd', 33}
s.remove('a')
s  # {'b', 'c', 'd', 33}
s.remove('banana')  # KeyError: 'banana'
s.discard('banana')  # None
s.pop('a')  # 'a'
```

Hashable

- Hashables – по большому счету все, кроме изменяемых коллекций

```
s.add([1, '2', 3])  # TypeError: unhashable type: 'list'
s.add((1, '2', 3))
s  # {'b', 'c', 'd', 33, (1, 2, 3)}
s.add((1, 2, [4, 5]))  # TypeError: unhashable type: 'list'

hash(s)  # TypeError: unhashable type: 'set'

fs = frozenset('asdf')
hash(s)  # -5967293086059881150
```

Еще методы для Set

```
s1 = set('asdf')
s2 = set('qwerty')

s1 > s2  # s1.superset(s2)
s1 | s2  # s1.union(s2)
s1 & s2  # s1.intersection(s2)
s1 - s2  # s1.difference(s2)

s1 == s2  # False
s1 > s2  # False
s1 < s2  # False
```

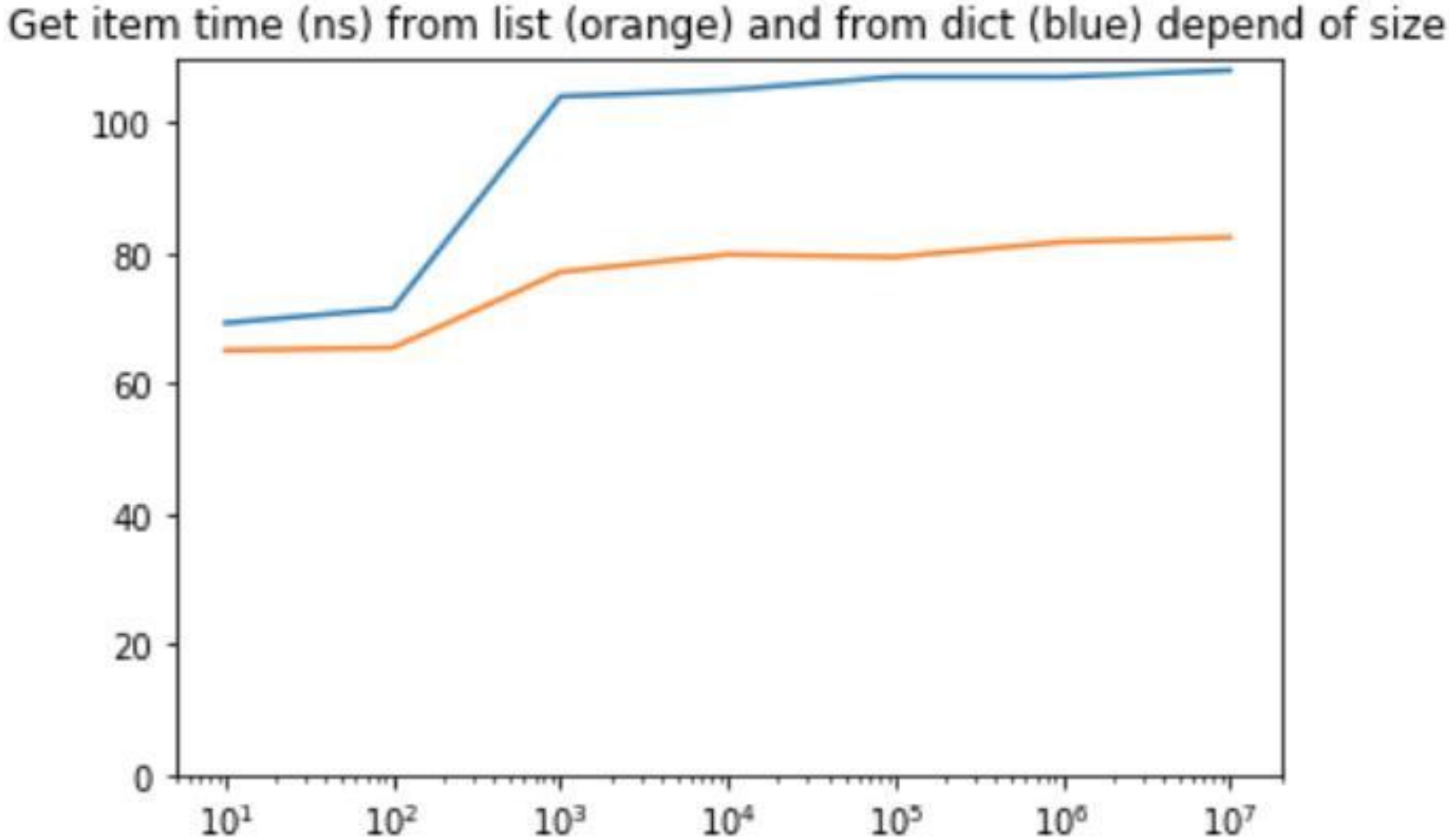
dict

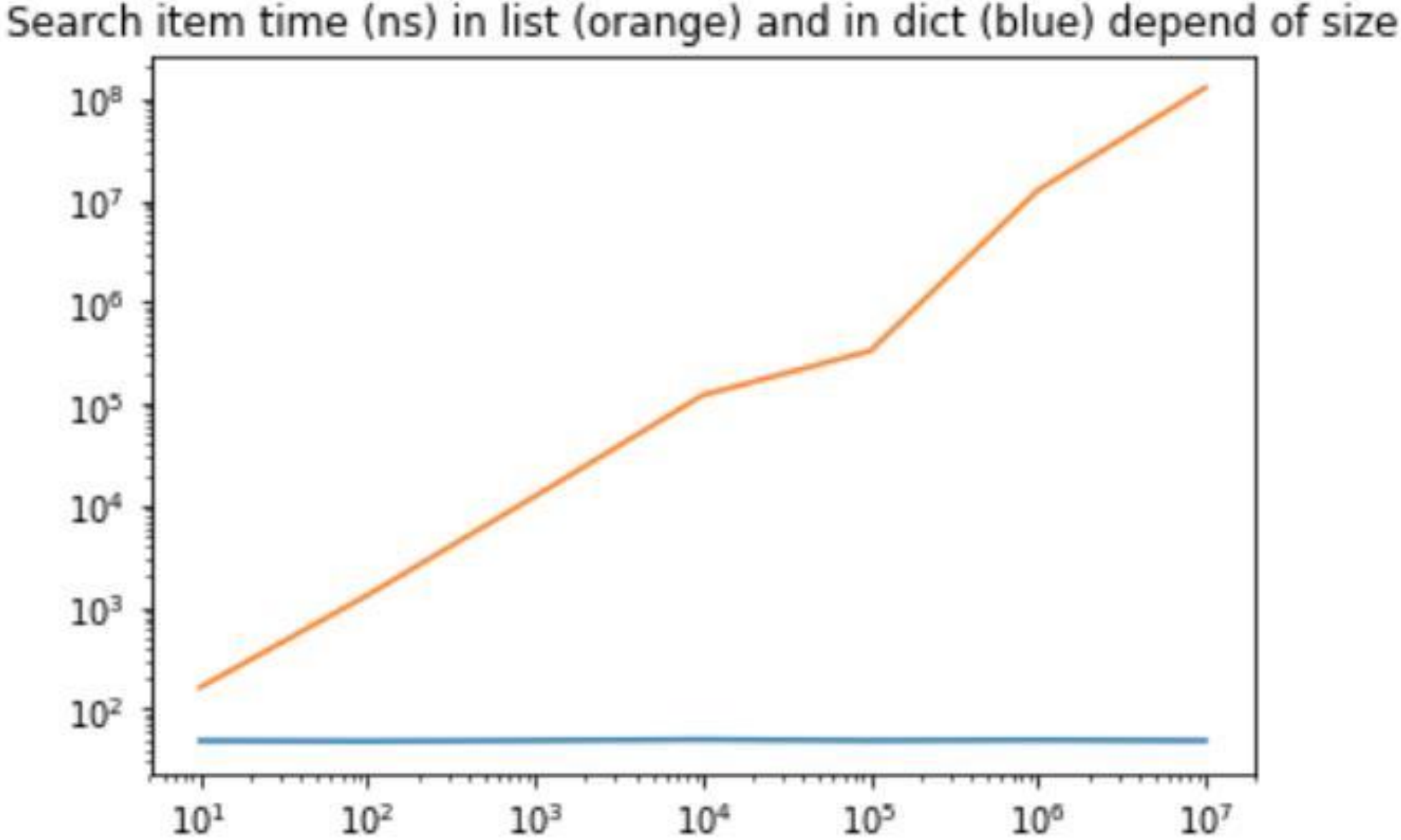
- Описывается парами ключ-значение
- Ключи – уникальные и хэшируемые
- Значения – произвольные

```
d1 = {'key1': 1, 'key2': 2}
d2 = dict(key1=1, key2=2)
d3 = dict((('key1', 1), ('key2', 2)))


d1 == d2 == d3  # True
```

Основные методы dict

```
d1['key1'] = 5
d1  # {'key1': 5, 'key2': 2}
d1['key3']  # KeyError 'key3'
d1.get('key3')  # None

d1.update({'key3': 3})
d1  # {'key1': 5, 'key2': 2, 'key3': 3}
d1.pop('key3')  # 3
d1  # {'key1': 5, 'key2': 2}
```


Get item time (ns) from list (orange) and from dict (blue) depend of size
100
80
60
40
20
0
10^1
10^2
10^3
10^4
10^5
10^6
10^7


Search item time (ns) in list (orange) and in dict (blue) depend of size
10^8
10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^2
10^3
10^4
10^5
10^6
10^7

Общее для коллекций и строк

```
len('many symbols')  # 12
len(set())  # 0

'd' in 'asdf'  # True
'Petya' in {'Name': 'Petya'}  # True
'Petya' not in {'Name': 'Petya'}  # False

for char in 'asdf':
    print(char)
# a
# s
...
```

Цикл for

```
lst = ['first', "don't print me", 'third', 'stop it', 'what about me? =(']

for item in lst:
    if item == "don't print me":
        continue
    if item == 'stop it':
        break
    print(item)
else:
    print('the cycle must not be broken')

# 'first'
# 'third'
```

Цикл for для dict

- По умолчанию итерируется по ключам

```
d = {'man': 'Rick', 'boy': 'Morty'}

for element in d:
    print(element)
# 'man'
# 'boy'

for key, value in d.items():
    print('the {} is {}'.format(key, value))
# the man is Rick
# the boy is Morty
```

Цикл for для dict

```
d = {'man': 'Rick', 'boy': 'Morty'}


for key, value in d.items():
    print('the {} is {}'.format(key, value))
    d['boy'] = 'Summer'
# the man is Rick
# the boy is Summer
```

Цикл for для dict

```
d = {'man': 'Rick', 'boy': 'Morty'}


for key, value in d.items():
    print('the {} is {}'.format(key, value))
    d['girl'] = 'Summer'
# the man is Rick
# RuntimeError: dictionary changed size during iteration
```

Цикл while

```
i = 0
while i < 5:
    if i == 4:
        i += 1
        continue
    if i == 5:
        break
    print(i)
    i += 1
else:
    print("now i'll be printed")
```

```
# 0
# 1
# 2
# 3
# now i'll be printed
```