Курс “Алгоритмы на python”

Занятие #3
Хеш-таблицы

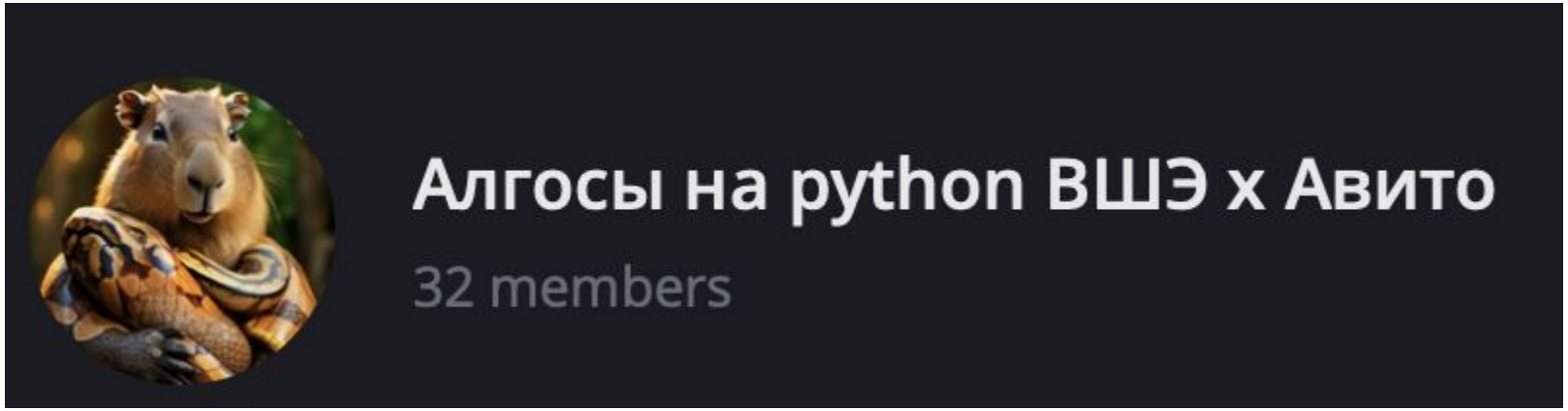
Сентябрь 2025

Avito

Единая точка входа/выхода – степик

- https://stepik.org/course/251189/

Вопросы и обсуждения – чат



Посещаемость

Орг моменты

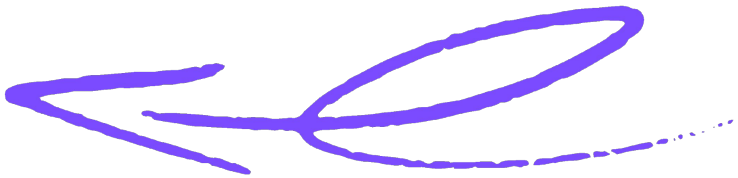
Структура курса «Алгоритмы на питоне»

1 модуль

- Введение в алгоритмы
- Базовые структуры данных
- Хеш-таблицы
- Кучи
- Бинарные деревья поиса
- Сортировки
- Рекурсия

2 модуль

- Графы
- Динамическое программирование
- Алгоритмы в строках
- Алгоритмы в ML и LLM
- Итоговый контест



План занятия

- Часть I. Изменяемые/неизменяемые типы в python
- Часть II. list, заглянем под капот
- Часть III. Хеш-таблицы
- Часть IV. Связный список с фиктивной нодой

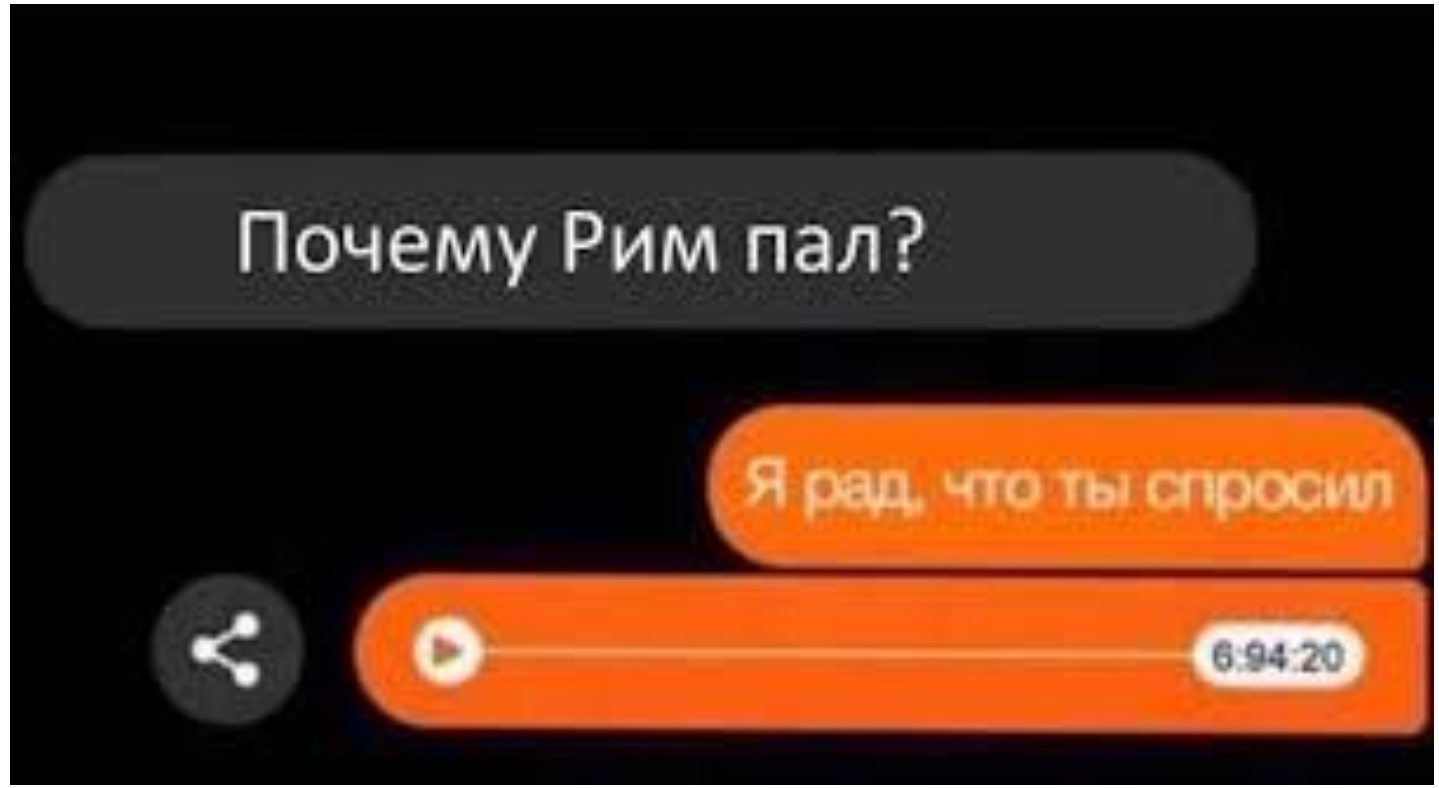
3 Хеш-таблицы

Типы данных в питоне

Все есть объект


Объекты
Неизменяемые
Изменяемые
числа
строки
кортежи
списки
словари
множества

Что в python значит изменяемость?


Почему Рим пал?
Я рад, что ты спросил
6:94:20

Связывание

```
>>>
>>> a = [1, 2, 3]
>>> b = a
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
>>> a is b
True
>>> c = [1, 2, 3]
>>> d = [1, 2, 3]
>>> c is d
False
>>> c == d
True
```

У каждого объекта есть id – по сути адрес в памяти

```
id(object, /)
```

Return the “identity” of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime. Two objects with non-overlapping lifetimes may have the same `id()` value.

CPython implementation detail: This is the address of the object in memory.

Примеры

```
>>> id(123)
4381700064
>>> id(999888)
4374579216
>>> id(123.45)
4374579280
>>> id('123.45')
4374754624
>>>
```

Операции сравнения
== vs is

is сравнивает идентификаторы

```
>>> a = [1,2]
>>> id(a)
4374836352
>>> b = [1,2]
>>> id(b)
4374836992
>>> a is b
False
```

== сравнивает значения

```
>>> a = [1,2]
>>> id(a)
4374836352
>>> b = [1,2]
>>> id(b)
4374836992
>>> a is b
False
>>> a == b
True
```

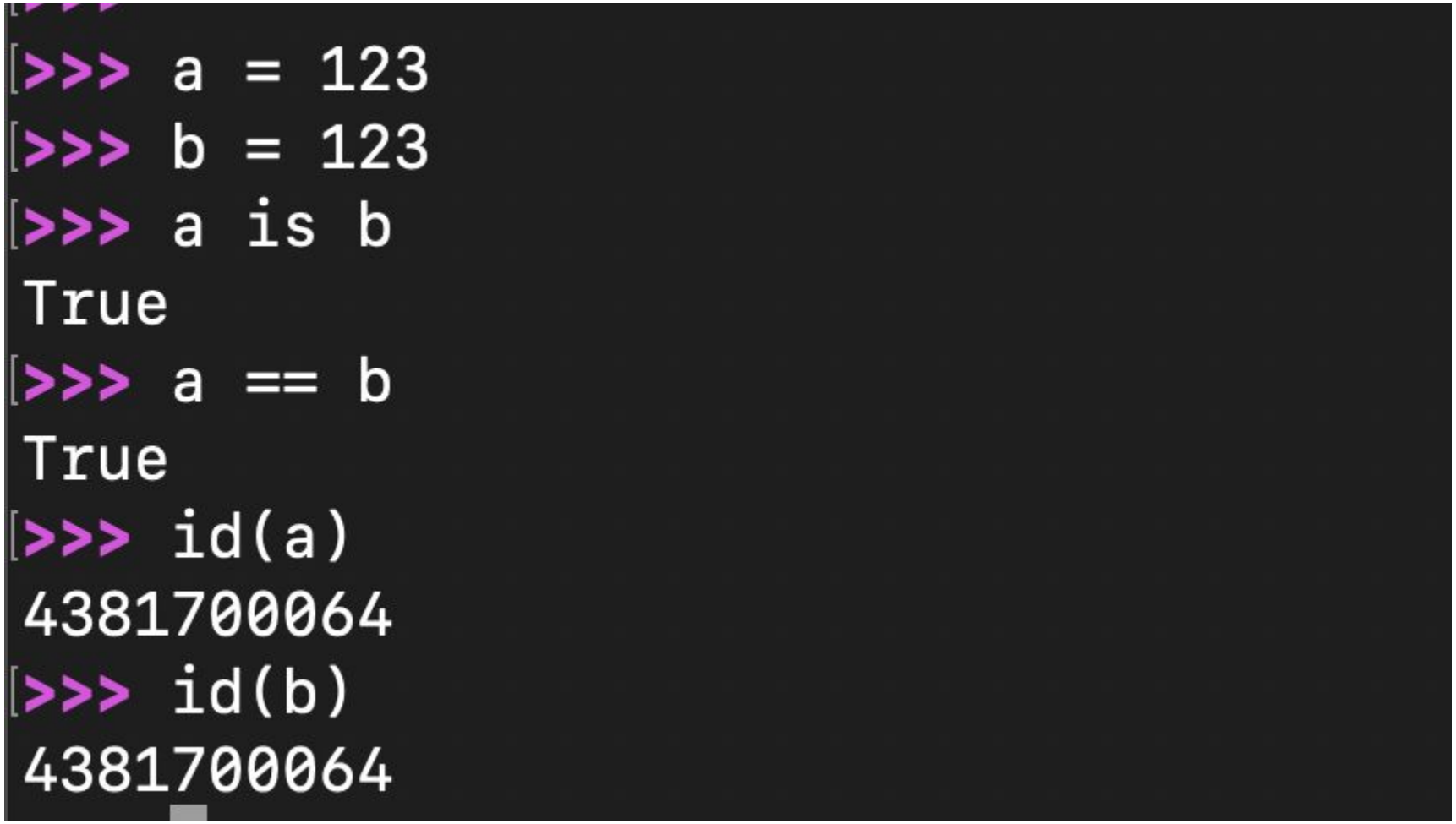
Разгребаем и убеждаемся

```
>>> a = [1, 2, 3]
>>> id(a)
4374805952
>>> sys.getrefcount(a)
2
>>> b = a
>>> id(b)
4374805952
>>> sys.getrefcount(a)
3
```

Разгребаем и убеждаемся

```
>>> c = [1, 2, 3]
>>> id(c)
4374799680
>>> sys.getrefcount(c)
2
>>> d = [1, 2, 3]
>>> id(d)
4374724544
>>> sys.getrefcount(c)
2
>>> sys.getrefcount(d)
2
```

Если все понятно, то, вот

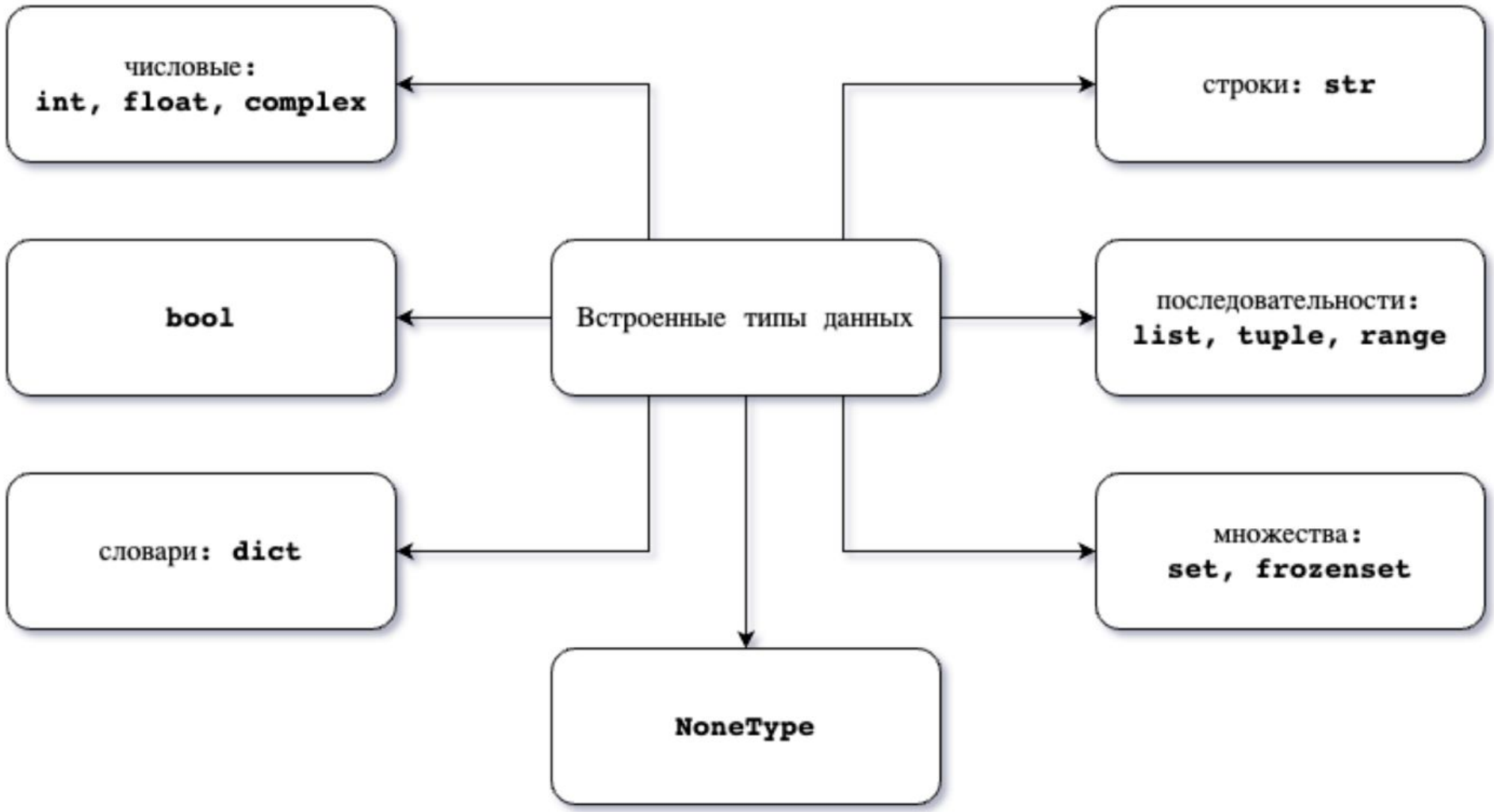


Числа от -5 до 256 создаются один раз при запуске интерпретатора и переиспользуются

Почему все-таки Рим пал?

```
>>> a = [1,2,3]
>>> id(a)
4374801856
>>> a[1] = 11
>>> a
[1, 11, 3]
>>> id(a)
4374801856
>>> tt = (1,2,3)
>>> tt[1] = 11
Traceback (most recent call last):
  File "<python-input-148>", line 1, in <module>
    tt[1] = 11
    ~~^^^
TypeError: 'tuple' object does not support item assignment
>>> type(tt)
<class 'tuple'>
```

Пробежимся по всем типам


числовые:
int, float, complex
строки: str
bool
Встроенные типы данных
последовательности:
list, tuple, range
словари: dict
множества:
set, frozenset
NoneType

```
156  struct _object {
157      // ob_tid stores the thread id (or zero). It is also used by the GC and the
158      // trashcan mechanism as a linked list pointer and by the GC to store the
159      // computed "gc_refs" refcount.
160      _Py_ALIGNED_DEF(_PyObject_MIN_ALIGNMENT, uintptr_t) ob_tid;
161      uint16_t ob_flags;
162      PyMutex ob_mutex;           // per-object lock
163      uint8_t ob_gc_bits;         // gc-related state
164      uint32_t ob_ref_local;      // local reference count
165      Py_ssize_t ob_ref_shared;   // shared (atomic) reference count
166      PyTypeObject *ob_type;
167  };
168  #endif // !defined(_Py_OPAQUE_PYOBJECT)
```

```
17
18      typedef struct _object PyObject;
19      typedef struct _longobject PyLongObject;
20      typedef struct _typeobject PyTypeObject;
```

int – PyLongObject

```
>>> type(123)
<class 'int'>
```

Посмотреть глазами исходники можно тут –
https://github.com/python/cpython/blob/main/Objects/longobject.c

кеш маленьких int-ов

```
26  #define medium_value(x) ((stwodigits)_PyLong_CompactValue(x))
27
28  #define IS_SMALL_INT(ival) (-_PY_NSMALLNEGINTS <= (ival) && (ival) < _PY_NSMALLPOSINTS)
```

есть конструктор int

```
>>> int(123.45)
123
>>> int('123')
123
```

bool

подтип int с фиксированными значениями True/False

```
>>> type(True)
<class 'bool'>
```

isinstance(*object*, *classinfo*, /)

Return `True` if the *object* argument is an instance of the *classinfo* argument, or of a (direct, indirect, or virtual) subclass thereof. If *object* is not an object of the given type, the function always returns `False`. If *classinfo* is a tuple of type objects (or recursively, other such tuples) or a Union Type of multiple types, return `True` if *object* is an instance of any of the types. If *classinfo* is not a type or tuple of types and such tuples, a `TypeError` exception is raised. `TypeError` may not be raised for an invalid type if an earlier check succeeds.

Доказательство

```
>>>
>>> isinstance(True, bool)
True
>>> isinstance(True, int)
True
```

NoneType – PyNone

None – единственный экземпляр типа NoneType

```
>>> type(None)
<class 'NoneType'>
```

float – хранится как C double внутри PyFloatObject

```
4
5     typedef struct {
6         PyObject_HEAD
7         double ob_fval;
8     } PyFloatObject;
```

```
>>> type(123.45)
<class 'float'>
```

Смотреть исходники тут
https://github.com/python/cpython/blob/main/Include/cpython/floatobject.h

посмотреть точность и диапазон

```
>>> print(sys.float_info)
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53,
epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

str – неизменяемый тип данных

```
>>> s = 'hello'
>>> c = 'hello'
>>> id(s)
4374841216
>>> id(c)
4374841216
>>> s is c
True
```

интернирование
тоже есть

```
>>> a = 'fun not fun test'
>>> c = 'fun not fun test'
>>> id(a)
4374859696
>>> id(c)
4374860272
>>> a is c
False
```

list и tuple

list – динамический массив указателей PyObject*

Смотреть исходники тут
https://github.com/python/cpython/blob/main/Objects/listobject.c

Разберемся-таки с динамичностью

```
>>> a = []
>>> a.__sizeof__()
40
>>> b = [1]
>>> b.__sizeof__()
48
```

Заголовок + динамический массив указателей на объекты

```
>>> header = [].__sizeof__()
>>> one = [1].__sizeof__() - header
>>> header, one
(40, 8)
```

Посчитаем количество элементов, под которые выделена память

```
>>> for length in range(30):
...     obj = [x for x in range(length)]
...     size = obj.__sizeof__()
...     allocated =  (size - header) // one
...     print('len = ', len(obj), ', capacity =', allocated)
...
len =  0 , capacity = 0
len =  1 , capacity = 4
len =  2 , capacity = 4
len =  3 , capacity = 4
len =  4 , capacity = 4
len =  5 , capacity = 8
len =  6 , capacity = 8
len =  7 , capacity = 8
len =  8 , capacity = 8
len =  9 , capacity = 16
len =  10 , capacity = 16
len =  11 , capacity = 16
len =  12 , capacity = 16
len =  13 , capacity = 16
len =  14 , capacity = 16
len =  15 , capacity = 16
len =  16 , capacity = 16
len =  17 , capacity = 24
len =  18 , capacity = 24
len =  19 , capacity = 24
len =  20 , capacity = 24
len =  21 , capacity = 24
len =  22 , capacity = 24
len =  23 , capacity = 24
len =  24 , capacity = 24
len =  25 , capacity = 32
len =  26 , capacity = 32
len =  27 , capacity = 32
len =  28 , capacity = 32
len =  29 , capacity = 32
```

если очень интересно, можно посмотреть реализацию list_resize тут https://github.com/python/cpython/blob/main/Objects/listobject.c#L108

tuple – кортеж – неизменяемый – фиксированный массив указателей

зачем надо?
работает быстрее
не надо ни о чем думать
может быть ключом словаря

tuple

```
>>> tt = (1,2,3)
>>> type(tt)
<class 'tuple'>
>>> tt[1] = 123
Traceback (most recent call last):
  File "<python-input-240>", line 1, in <module>
    tt[1] = 123
    ~~^^^
TypeError: 'tuple' object does not support item assignment
```

tuple

```
>>> ll = [1,2,3]
>>> tt = (1, 2, ll)
>>> tt
(1, 2, [1, 2, 3])
>>> ll.append(123)
>>> tt
(1, 2, [1, 2, 3, 123])
>>>
```

Наконец, хеш-таблицы

под капотом у dict и set

Что такое хеш-таблица?

Какую структуру хотим?

- ассоциативный массив, хранящий пару (ключ, значение)
- эффективный поиск по ключу
- менять размер, не теряя в производительности

Рисуем на доске

Рисуем на доске

capacity
load factor
хеш-функция

Что такое хорошая хеш-функция?

Что такое хорошая хеш-функция?

- однозначность
- быстрое вычисление
- находиться в диапазоне массива
- равномерное распределение

Коллизии. Рисуем на доске

Метод цепочек. Рисуем на доске

Идея: хранить связный список, при поиске проходить по нему, сравнивая ключи

Открытая адресация. Рисуем на доске

Идея: искать следующую свободную ячейку и писать в нее

Открытая адресация. Рисуем на доске

линейное пробирование

$h(k) + j$

квадратичное пробирование

$h(k) + j^2$, $h(k) + j(j+1)/2$, $h(k) + j + j^2$

два хеша

$h(k) + j\ g(k)$

Открытая адресация. Рисуем на доске

не забываем mod m

Открытая адресация. Рисуем на доске

при открытой адресации удаление нельзя делать null – иначе прервется probe-последовательность

используют dummy

Как оно в python?

1) load factor – степень заполненности – 2/3

```
530
531     /* USABLE_FRACTION is the maximum dictionary load.
532      * Increasing this ratio makes dictionaries more dense resulting in more
533      * collisions.  Decreasing it improves sparseness at the expense of spreading
534      * indices over more cache lines and at the cost of total memory consumed.
535      *
536      * USABLE_FRACTION must obey the following:
537      *     (0 < USABLE_FRACTION(n) < n) for all n >= 2
538      *
539      * USABLE_FRACTION should be quick to calculate.
540      * Fractions around 1/2 to 2/3 seem to work well in practice.
541      */
542     #define USABLE_FRACTION(n) (((n) << 1)/3)
```

Как оно в python?

2) коллизии решаются пробированием

```
127             perturb >>= PERTURB_SHIFT;
128             i = (i * 5 + 1 + perturb) & mask;
```