Введение в Python

Лекция 6: Списки в Python

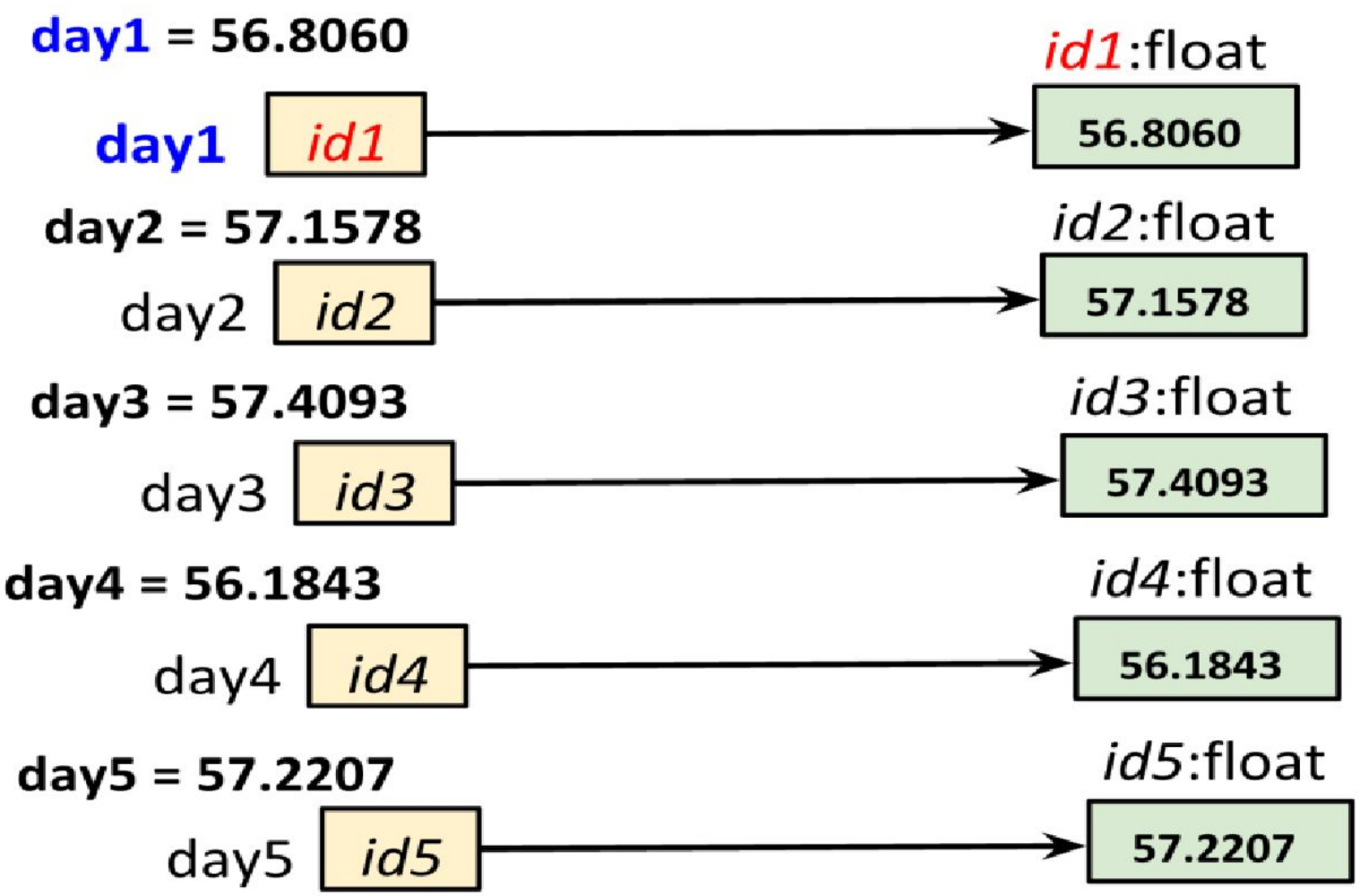
Создание списка. Пример:

Дата	EUR
16.04.2019	56.9881
15.04.2019	57.1383
13.04.2019	55.7388
11.04.2019	57.1102
5.04.2019	56. 8971
29.03.2019	57.2207
18.03.2019	56.1843
12.03.2019	57.4093
10.03.2019	57.1578

Поместим курс валют на каждый день в отдельную переменную:

```
>>> day1 = 56.8060
>>> day2 = 57.1578
>>>
```

Схематично:

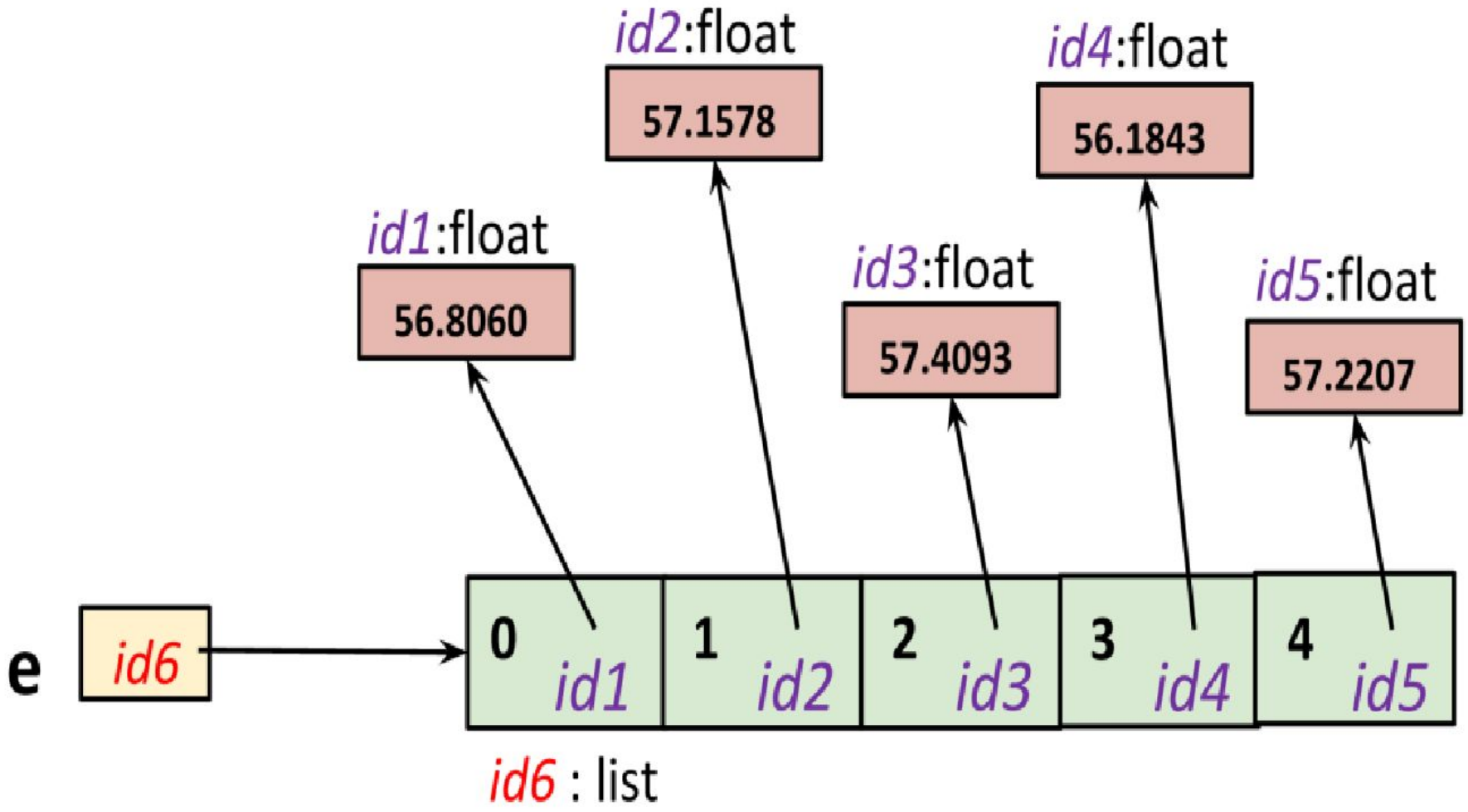

day1 = 56.8060
day1
id1
id1:float
56.8060
day2 = 57.1578
day2
id2
id2:float
57.1578
day3 = 57.4093
day3
id3
id3:float
57.4093
day4 = 56.1843
day4
id4
id4:float
56.1843
day5 = 57.2207
day5
id5
id5:float
57.2207

- Список (list) – это структура данных для хранения объектов различных типов.
- Список очень похож на массив, только в нем можно хранить объекты различных типов.
- Размер списка не статичен, его можно изменять.
- Список по своей природе является изменяемым типом данных.
- Переменная, определяемая как список, содержит ссылку на структуру в памяти, которая в свою очередь хранит ссылки на какие-либо другие объекты или структуры.

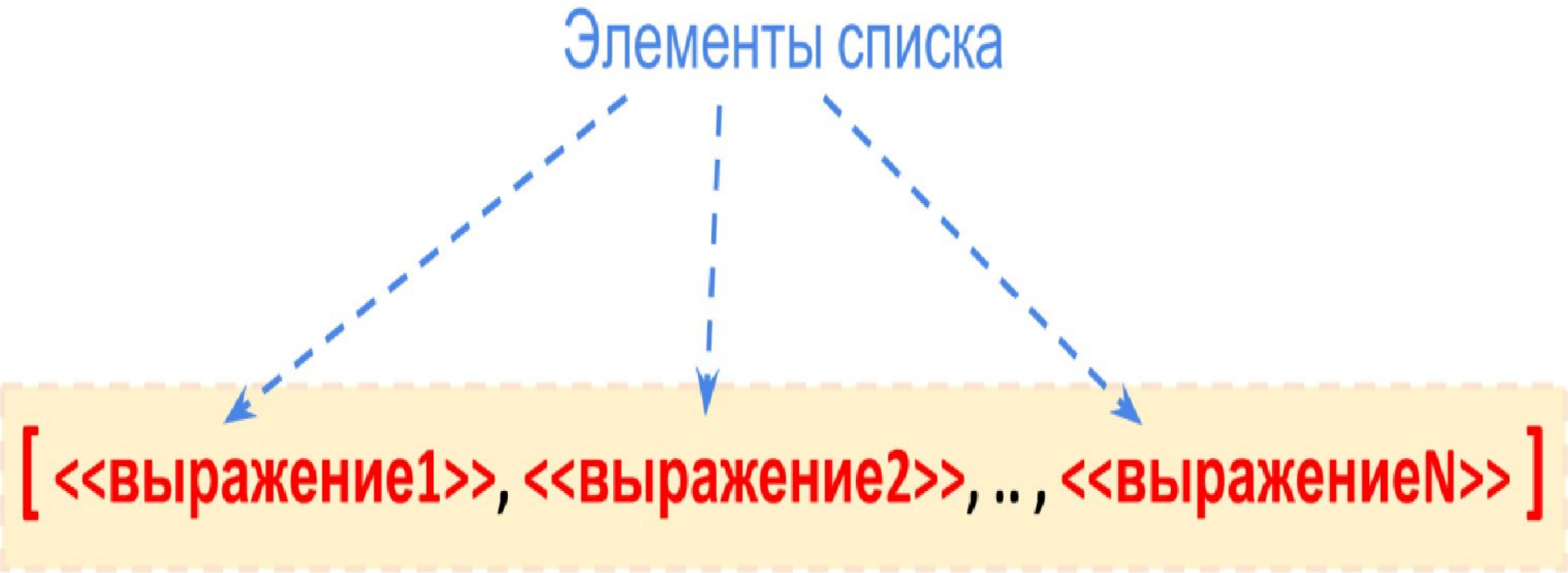
- Список (list) в Python является объектом, поэтому может быть присвоен переменной (переменная, как и в предыдущих случая, хранит адрес объекта класса список).
- Представим список с курсом валют:

```
>>> e = [56.8060, 57.1578, 57.4093, 56.1843,
57.2207]
>>> e
[56.806, 57.1578, 57.4093, 56.1843, 57.2207]
>>>
```

Как Python работает со списками в памяти:



В общем виде создание списка выглядит следующим образом:



Операции над списками:

Обращаться к отдельным элементам списка можно по их индексу (позиции), начиная с нуля:

```
>>> e=[56.8060, 57.1578, 57.4093, 56.1843, 57.2207]
>>> e[0]
56.806
>>> e[1]
57.1578
>>> e[-1] # последний элемент
57.2207
>>>
```

Обращение по несуществующему индексу вызовет ошибку:

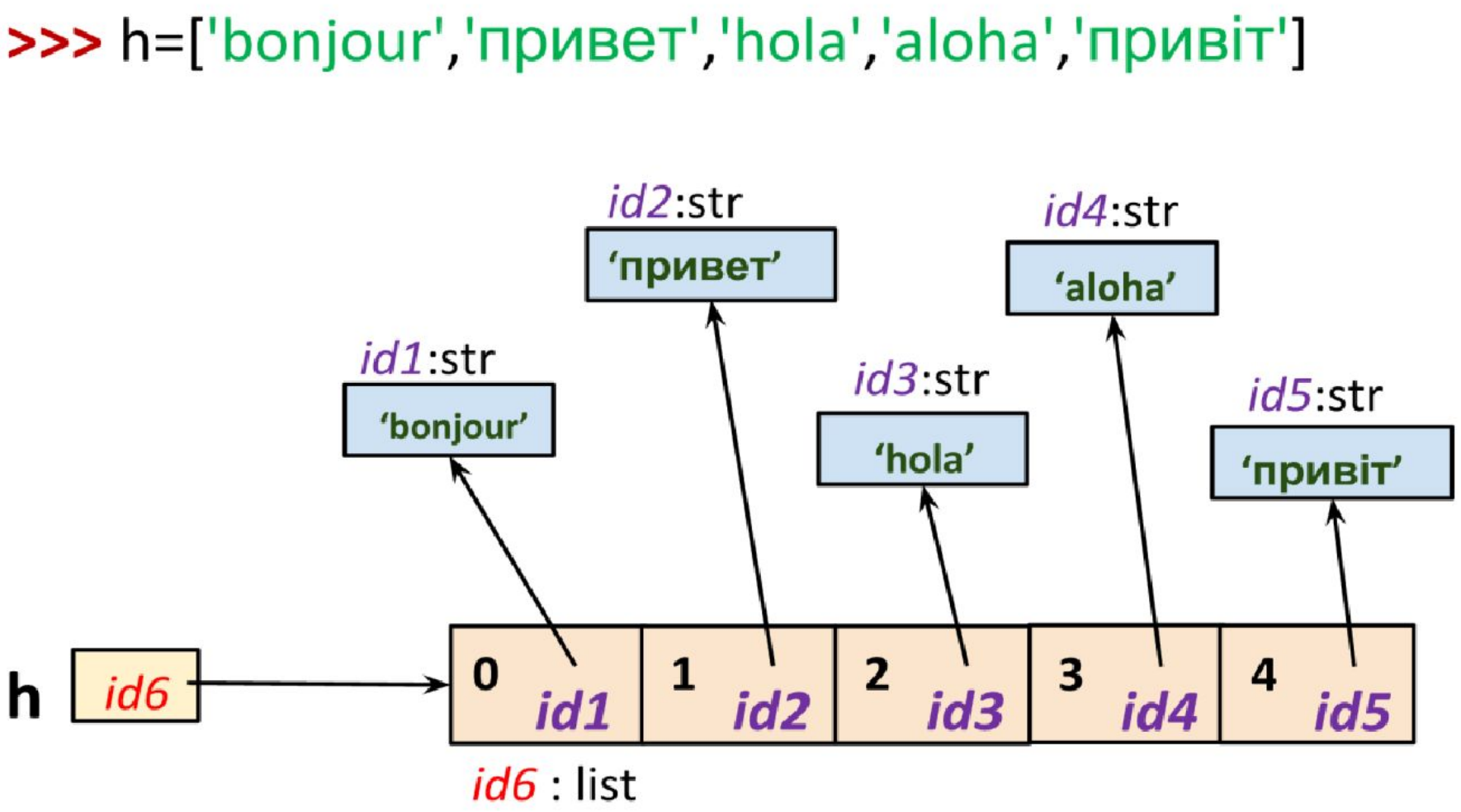
```
>>> e[100]
Traceback (most recent call last):
File "<pyshell#10>", line 1, in <module>
e[100]
IndexError: list index out of range
>>>
```

Списки можно изменить. Пример:

```
>>> h=['Hi', 27 , -8.1, [1,2]]
>>> h[1]='hello'
>>> h
['Hi', 'hello', -8.1, [1, 2]]
>>> h[1]
'hello'
>>>
```

Пример:

```
>>> h=['bonjour','привет','hola','aloha','привіт']
>>>
```

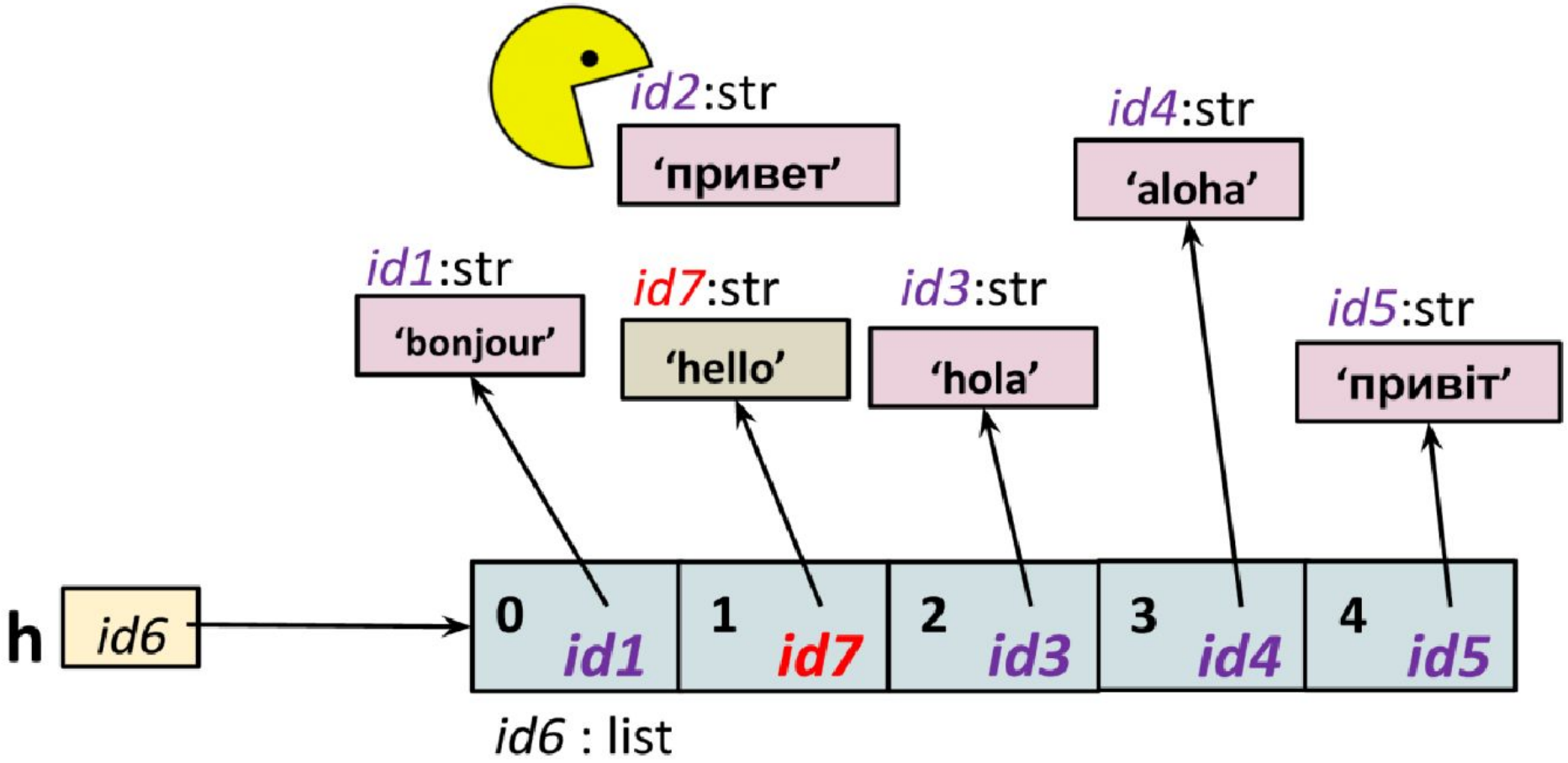


Производим изменения списка:

```
>>> h[1]='hello'
>>> h
['bonjour', 'hello', 'hola', 'aloha', 'привіт']
>>> h[1]
'hello'
>>>
```

В памяти:

```
>>> h[1]='hello'
```



- Список (`list`), наверное, наиболее часто встречающийся тип данных, с которым приходится сталкиваться при написании программ.
- Это связано со встроенными в Python функциями, которые позволяют легко и быстро обрабатывать списки:

Встроенные функции для обработки списков:

- `len(L)` – возвращает число элементов в списке L
- `max(L)` – возвращает максимальное значение в списке L
- `min(L)` – возвращает минимальное значение в списке L
- `sum(L)` – возвращает сумму значений в списке L
- `sorted(L)` – возвращает копию списка L, в котором элементы упорядочены по возрастанию. Не изменяет список L

Примеры вызовов функций:

```
>>> e=[56.8060, 57.1578, 57.4093, 56.1843, 57.2207]
>>> e
[56.806, 57.1578, 57.4093, 56.1843, 57.2207]
>>> len(e)
5
>>> max(e)
57.4093
>>> min(e)
56.1843
>>> sum(e)
284.7781
>>> sorted(e)
[56.1843, 56.806, 57.1578, 57.2207, 57.4093]
>>> e
[56.806, 57.1578, 57.4093, 56.1843, 57.2207]
>>>
```

Упражнения:

`Дан список L = [3, 6, 7, 4, -5, 4, 3, -1]`

1. Определите сумму элементов списка `L`. ЕСЛИ сумма превышает значение `2`, ТО вывести на экран число элементов списка.

2. Определить разность между минимальным и максимальным элементами списка. ЕСЛИ абсолютное значение разности больше `10`, ТО вывести на экран отсортированный по возрастанию список, ИНАЧЕ вывести на экран фразу «`Разность меньше 10`».

Операция + для списков служит для их объединения:

```
>>> original=['H','B']
>>> final=original+['T']
>>> final
['H', 'B', 'T']
```

Операция повторения:

```
>>> final=final*5
>>> final
['H', 'B', 'T', 'H', 'B', 'T', 'H', 'B', 'T',
'H', 'B', 'T', 'H', 'B', 'T']
```

Инструкция `del` позволяет удалять из списка элементы по индексу:

```
>>> del final[0]
>>> final
['B', 'T', 'H', 'B', 'T', 'H', 'B', 'T', 'H',
'B', 'T', 'H', 'B', 'T']
```

Пример:

Напишем функцию, объединяющую два списка:

```
>>> def f(x,y):
return x+y
>>> f([1,2,3],[4,5,6])
[1, 2, 3, 4, 5, 6]
>>>
```

Теперь передадим в качестве аргументов две строки:

```
>>> f("123", "456")
'123456'
>>>
```

Передадим два числа:

```
>>> f(1,2)
3
```

- Таким образом, получилась небольшая функция, которая может объединять и складывать в зависимости от класса (типа данных) переданных ей объектов.

Следующий полезный оператор `in`:

```
>>> h=['bonjour',7,'hola',-1.0,'привіт']
>>> if 7 in h:
print ('Значение есть в списке')
Значение есть в списке
>>>
```

Упражнение:

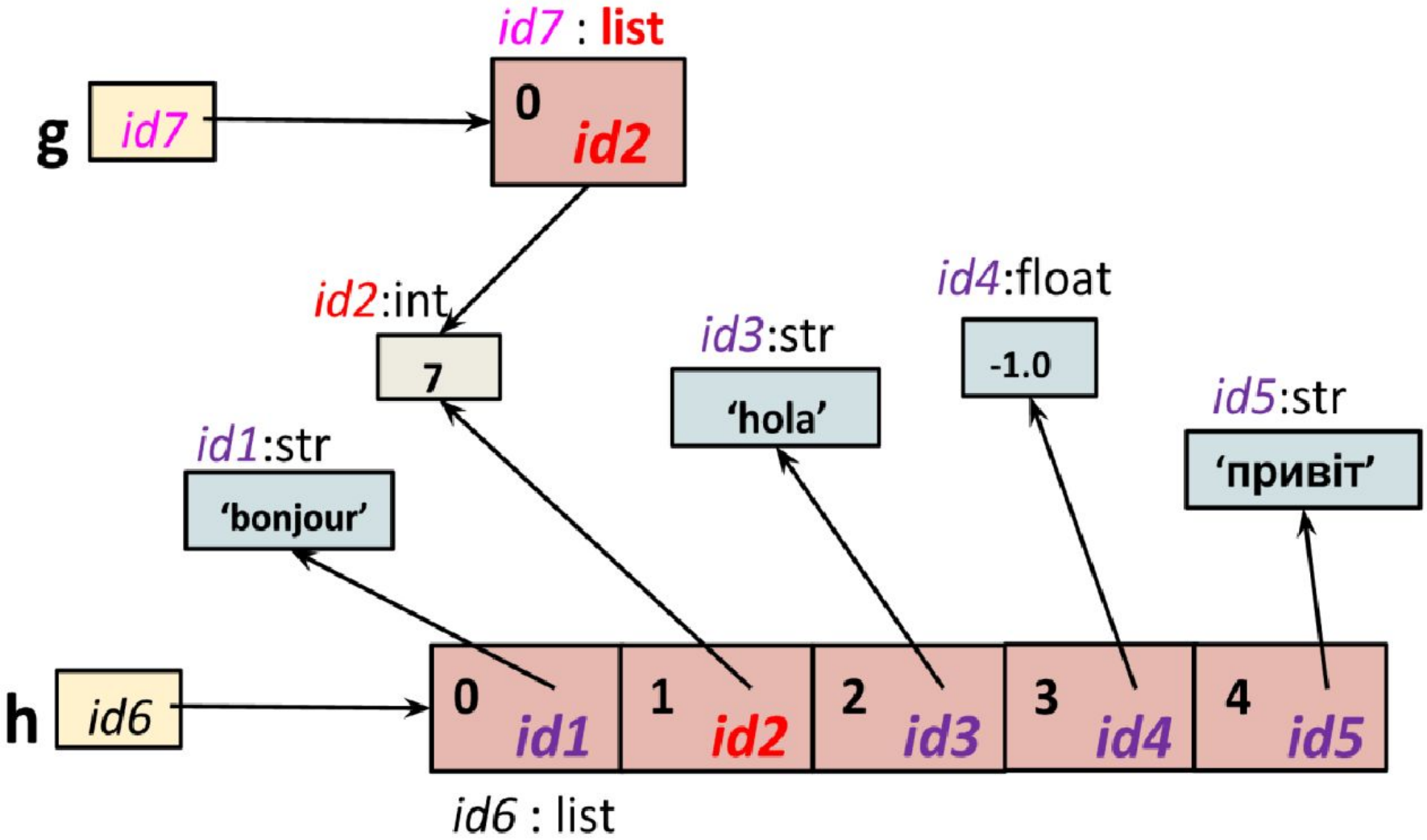
```
Дан список: L = [3, 'hello', 7, 4, 'привет', 4,
3, -1]
```

Определите наличие строки «привет» в списке. ЕСЛИ такая строка в списке присутствует, ТО вывести ее на экран, повторив 10 раз.

Для списка также есть операция взятия среза:

```
>>> h=['bonjour',7,'hola',-1.0,'привіт']
>>> h
['bonjour', 7, 'hola', -1.0, 'привіт']
>>> g=h[1:2]
>>> g
[7]
>>>
```

В памяти это выглядит следующим образом:



Вернемся к инструкции `del` и удалим с помощью среза подсписок:

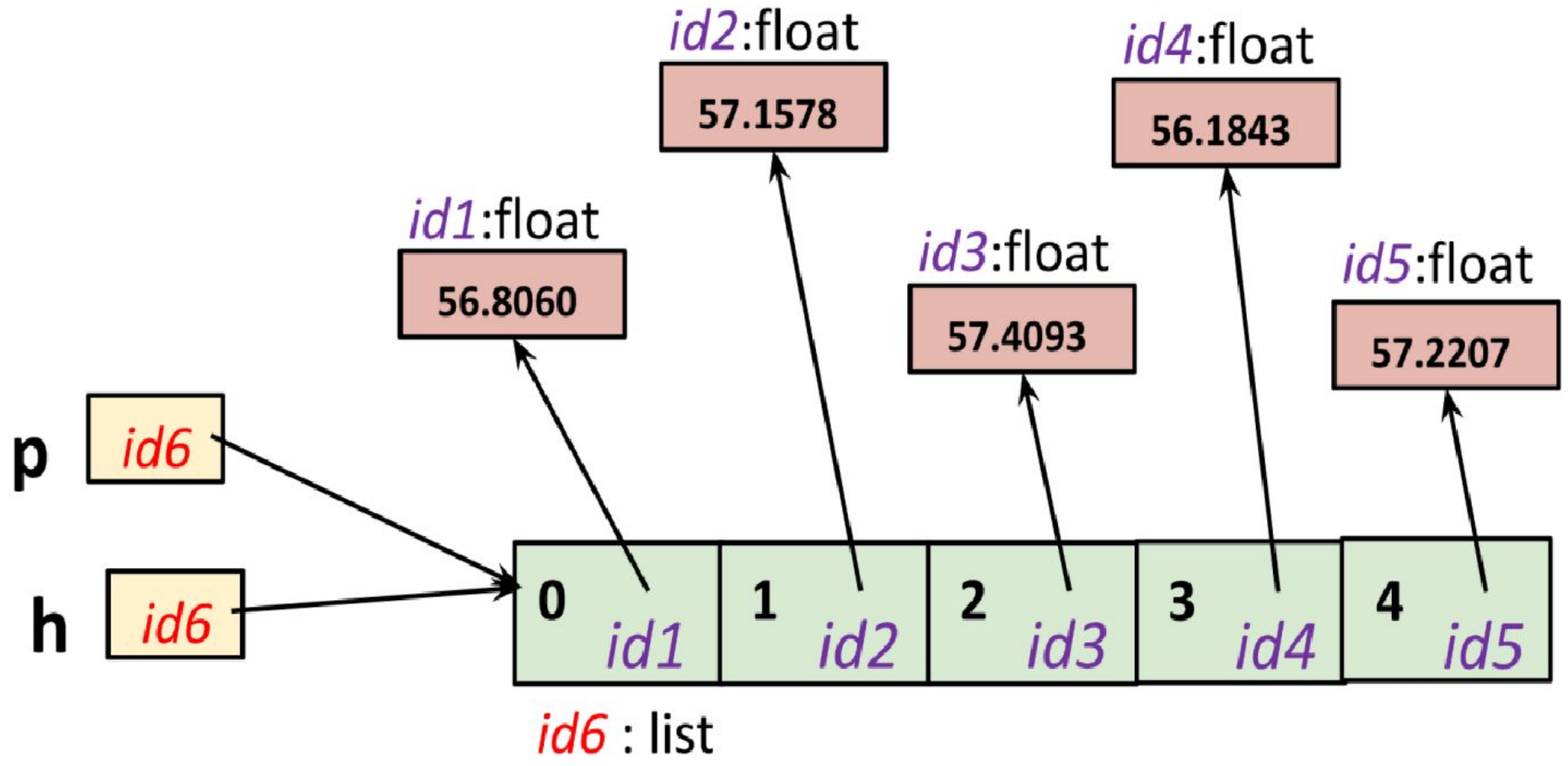
```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4] # удаление подсписка
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
>>>
```

Псевдонимы и копирование списков

Выполним следующий код:

```
>>> h
['bonjour', 7, 'hola', -1.0, 'привіт']
>>> p=h # содержат указатель на один и тот же список
>>> p
['bonjour', 7, 'hola', -1.0, 'привіт']
>>> p[0]=1 # модифицируем одну из переменных
>>> h # изменилась другая переменная!
[1, 7, 'hola', -1.0, 'привіт']
>>> p
[1, 7, 'hola', -1.0, 'привіт']
>>>
```

В Python две переменные называются *псевдонимами*, когда они содержат одинаковые адреса памяти.



Как проверить, ссылаются ли переменные на один и тот же список:

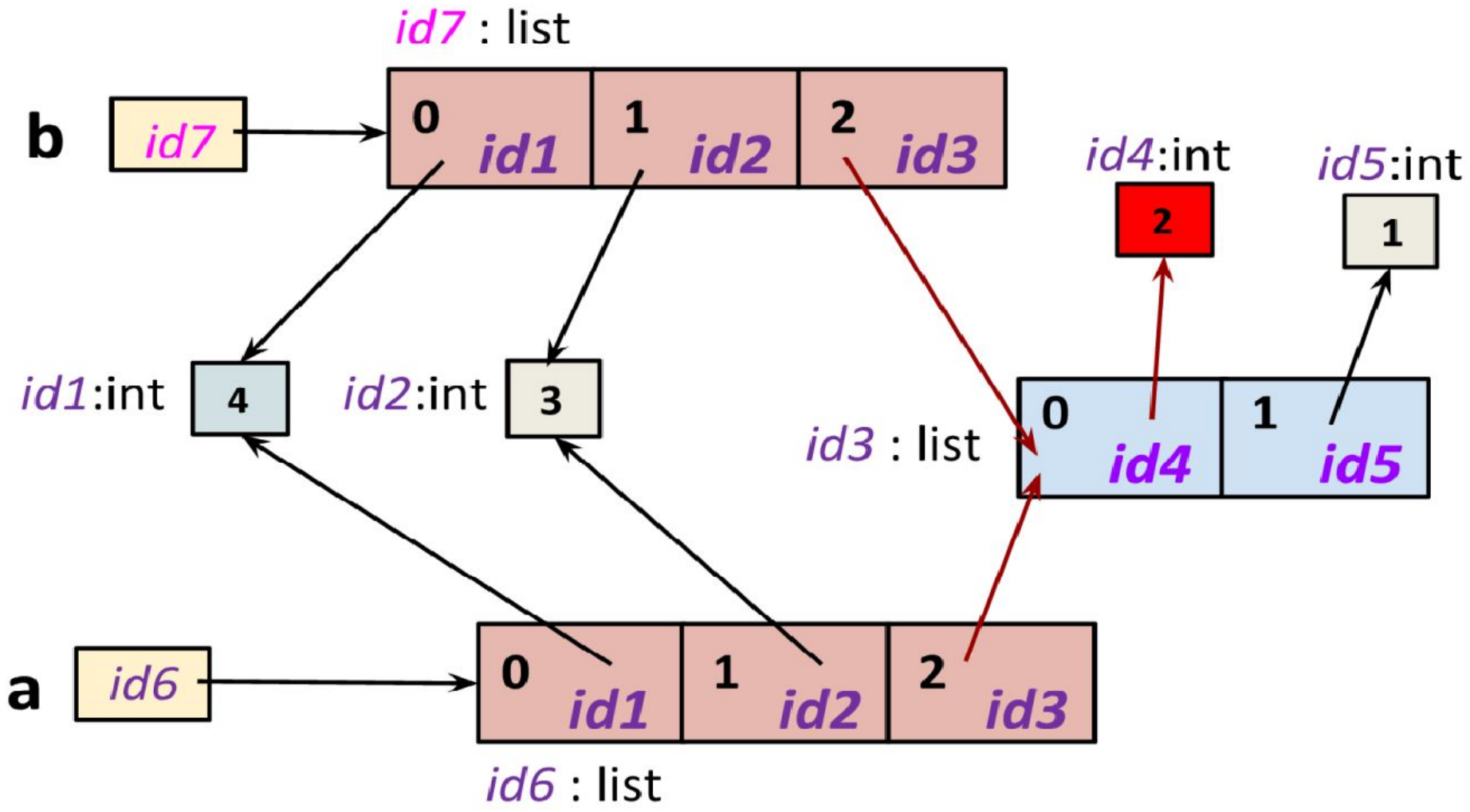
```
>>> x = y = [1, 2] # создали псевдонимы
>>> x is y # проверка, ссылаются ли переменные на один и тот же объект45
True
>>> x = [1, 2]
>>> y = [1, 2]
>>> x is y
False
>>>
```

К спискам применимы два вида копирования

Первый вид – *поверхностное копирование*, при котором создается новый объект, но он будет заполнен ссылками на элементы, которые содержались в оригинале:

```
>>> a = [4, 3, [2, 1]]
>>> b = a[:]
>>> b is a
False
>>> b[2][0]=-100
>>> a
[4, 3, [-100, 1]] # список a тоже изменился
>>>
```

Схема размещения ссылок на объекты при поверхностном копировании:



Второй вид копирования – *глубокое копирование*

При глубоком копировании создается новый объект и рекурсивно создаются копии всех объектов, содержащихся в оригинале:

```
>>> import copy
>>> a = [ 4, 3, [2, 1] ]
>>> b = copy.deepcopy(a)
>>> b[2][0]=-100
>>> a
[4, 3, [2, 1]] # список a не изменился
>>>
```

Методы списка

```
>>> colors=['red', 'orange', 'green']
>>> colors.extend(['black','blue']) #  расширяет список списком
>>> colors
['red', 'orange', 'green', 'black', 'blue']
>>> colors.append('purple') # добавляет элемент в список
>>> colors
['red', 'orange', 'green', 'black', 'blue', 'purple']
>>> colors.insert(2,'yellow') # добавляет элемент в указанную позицию
>>> colors
['red', 'orange', 'yellow', 'green', 'black', 'blue', 'purple']
>>> colors.remove('black') # удаляет элемент из списка
>>> colors
['red', 'orange', 'yellow', 'green', 'blue', 'purple']
>>> colors.count('red') # считает количество повторений аргумента метода
1
>>> colors.index('green') # возвращает позицию в списке аргумента метода
3
```

Еще несколько полезных методов списка:

```
>>> colors
['red', 'orange', 'yellow', 'green', 'blue', 'purple']
>>> colors.pop() # удаляет и возвращает последний элемент списка
'purple'
>>> colors
['red', 'orange', 'yellow', 'green', 'blue']
>>> colors.reverse() # список в обратном порядке
>>> colors
['blue', 'green', 'yellow', 'orange', 'red']
>>> colors.sort() # сортирует список (вспомните о сравнении строк)
>>> colors
['blue', 'green', 'orange', 'red', 'yellow']
>>> colors.clear() # очищает список. Метод появился в версии 3.3. Аналог del color[:]
>>> colors
[]
>>>
```

Преобразование типов

- Очень часто появляется потребность в изменении строк, но напрямую мы этого сделать не можем. Тогда нам на помощь приходят списки. Преобразуем строку в список, изменим список, затем вернем его в строку:

```
>>> s='Строка для изменения'
>>> list(s)  # функция list() пытается преобразовать аргумент в список
['С', 'т', 'р', 'о', 'к', 'а', ' ', 'д', 'л', 'я', ' ',
'и', 'з', 'м', 'е', 'н', 'е', 'н', 'и', 'я']
>>> lst = list(s)
>>> lst[0]='М'  # изменяем список, полученный из строки
>>> lst
['М', 'т', 'р', 'о', 'к', 'а', ' ', 'д', 'л', 'я', ' ',
'и', 'з', 'м', 'е', 'н', 'е', 'н', 'и', 'я']
>>> s=''.join(lst)  # преобразуем список в строку с помощью строкового метода join()
>>> s
'Мтрока для изменения'
>>>
```

Рассмотрим несколько примеров строкового метода `join()`:

```
>>> A = ['red', 'green', 'blue']
>>> ' '.join(A)
'red green blue'
>>> ''.join(A)
'redgreenblue'
>>> '***'.join(A)
'red***green***blue'
>>>
```

Аналогично можно преобразовать число к списку (через строку) и затем изменить полученный список:

```
>>> n=734853847538465384 65
>>> list(str(n)) # число преобразуем в строку, затем строку в список
['7', '3', '4', '8', '5', '3', '8', '4', '7', '5', '3', '8', '4', '6', '5', '3', '8', '4', '6', '5']
>>>
```

Если строка содержит разделитель, то ее можно преобразовать к списку с помощью строкового метода `split()`, который по умолчанию в качестве разделителя использует пробел:

```
>>> s='d a dd dd gg rr tt yy rr ee'.split()
>>> s
['d', 'a', 'dd', 'dd', 'gg', 'rr', 'tt', 'yy',
'rr', 'ee']
>>>
```

Возьмем другой разделитель:

```
>>> s='d:a:dd:dd:gg:rr:tt:yy:rr:ee'.split(":")
>>> s
['d', 'a', 'dd', 'dd', 'gg', 'rr', 'tt', 'yy',
'rr', 'ee']
>>>
```

Вложенные списки

В качестве элементов списка могут быть объекты любого типа, например, списки:

```
>>> lst=[['A', 1], ['B',2], ['C',3]]
>>> lst
[['A', 1], ['B', 2], ['C', 3]]
>>> lst[0]
['A', 1]
>>>
```

- Подобные структуры используются для хранения матриц.
- Обращение (изменение) к вложенному списку происходит через указание двух индексов:

```
>>> lst[0][1]
1
>>>
```

Схематично вложенные списки выглядят следующим образом: