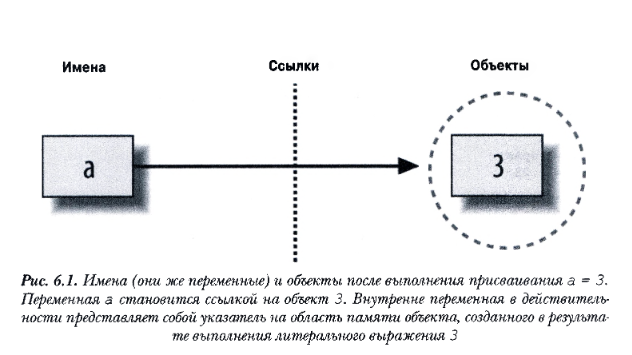
**21.5 Передача данных в функцию**

**А что такое переменные?**

Перед тем как приступить к рассмотрению изменяемых и неизменяемых типов данных, стоит вникнуть в работу Python с переменными, которая выполняется «под капотом».

Для начала возможный шок-контент: если мы напишем x = 5, то это совершенно не значит, что х хранит в себе число 5.

Число 5 «физически» хранится на диске, а переменная с названием x содержит только ссылку на ту область памяти, в которой хранится это число. Например, число 5 записано в ячейку памяти №123.



Источник: Марк Лутц «Изучаем Python»

Если мы напишем   
x = 5  
и   
y = x,  
то и х и y будут хранить адрес ячейки памяти №123, и, только пройдя по этой ссылке,  Python сможет узнать информацию о типе объекта и его содержании.

Так, например, происходит, когда мы пытаемся выполнить над переменной действие, которое нельзя выполнить с объектом, на который ссылается переменная.

Например, нельзя вызвать число как функцию:  
x = 5  
x() — Python даст выполнить эту строку.  
TypeError: 'int' object is not callable — но при выполнении вызовет ошибку.

Кстати, если мы изменим переменную x после этого:  
x = 10,  
то переменная y продолжит ссылаться на число 5,  
а переменная x будет ссылаться на число 10,  
то есть, когда мы пишем y = x, это значит, что мы добавляем в переменную y такую же ссылку, какая была в переменной x.  
Но дальше эти переменные не будут зависеть друг от друга.

Из этого всего стоит запомнить три понятия:

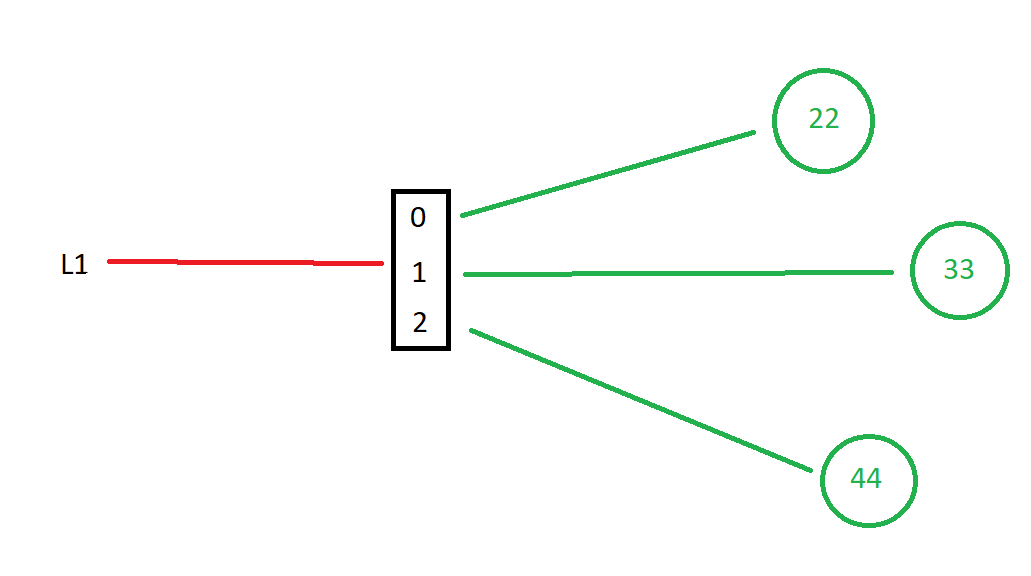
* переменная — по сути, это имя,
* объекты — это области памяти для хранения значений,
* ссылки — указатели от имени к области памяти.

**Работа с объектами. Начало**

Что происходит с объектами посложнее?

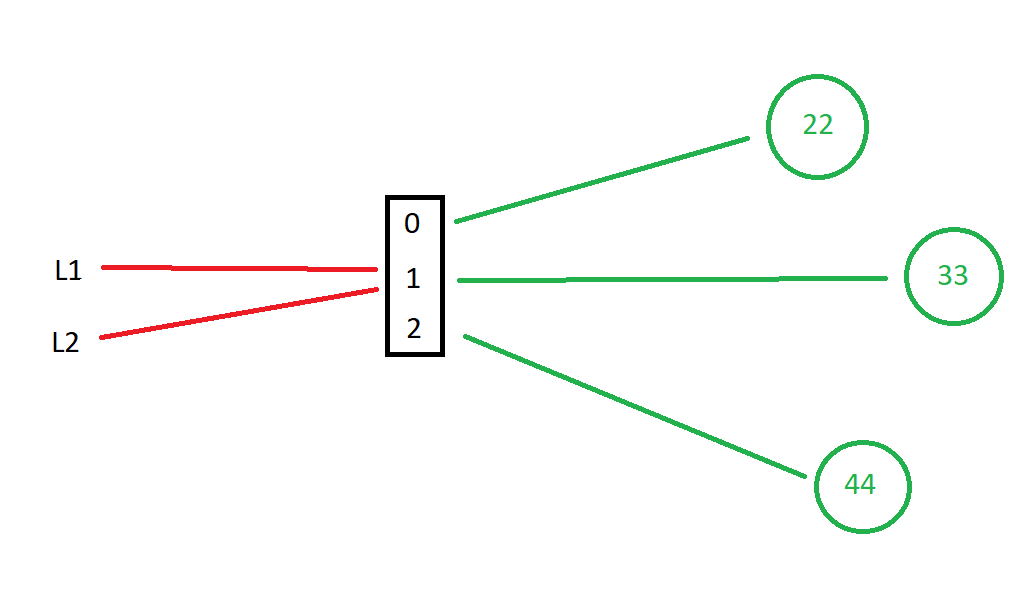
L1 = [22, 33, 44]

L1 будет именем, которое содержит указатель на область памяти со списком,  
но числа внутри списка — это тоже отдельные объекты, которые хранятся в своих ячейках.



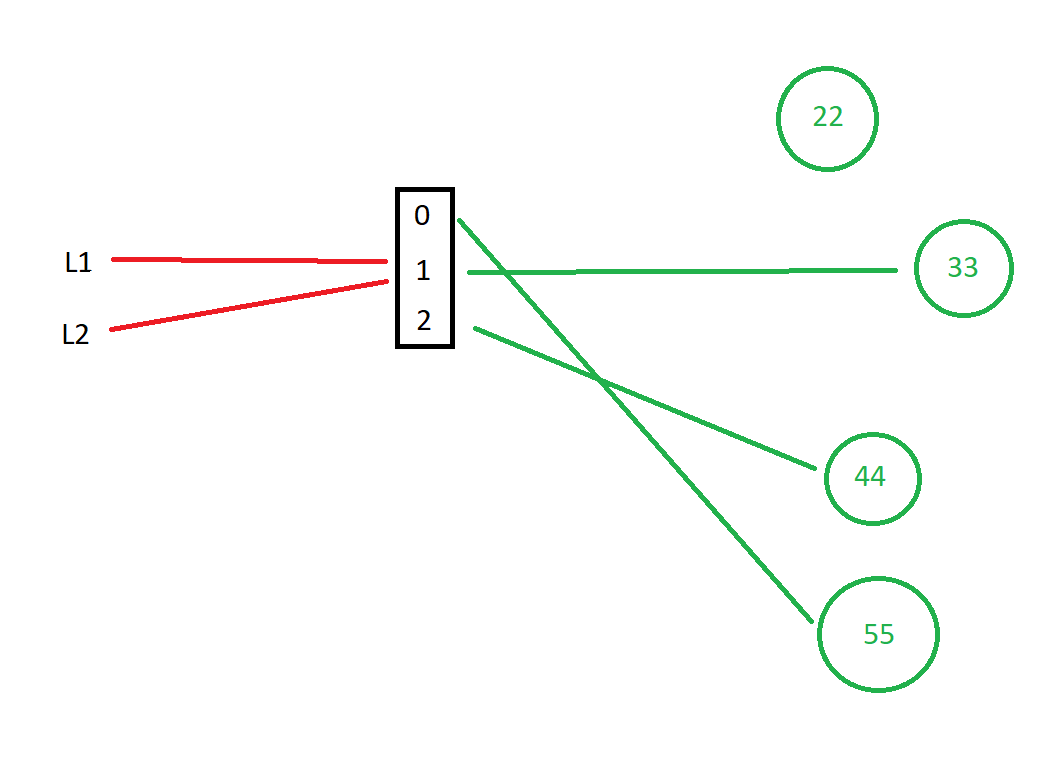
Автор Максим Л.В

Если мы напишем  
L2 = L1,  
то создадим ещё одно имя, которое будет ссылаться на этот же список.



Автор Максим Л.В

Если мы напишем  
L1[0] = 55,  
то, по сути, мы изменим ссылку на объект, который был внутри списка на первом месте.



Автор Максим Л.В

Если мы проверим после этого, что хранится в L2, то увидим те же самые числа:  
[55, 33, 44],  
так как L1[0] = 55 изменило не сам список, а объект внутри списка.

Такое поведение характерно для изменяемых объектов вроде списков или словарей. Это их особенность, которой можно и нужно пользоваться. Но правильно её использовать можно, только если вы знаете, как оно работает изнутри.

**Работа с объектами. Продолжение**

С неизменяемыми объектами сделать так не получится.

Например:  
x = (1, 2, 3)  
x[0] = 5 — попытка изменения кортежа приведёт к ошибке:

TypeError: 'tuple' object does not support item assignment

Но это не значит, что неизменяемые объекты хуже. Так как они не поддерживают такой довольно сложный механизм, они затрачивают меньше памяти на хранение тех же самых данных.

import sys

test\_list = list()  
test\_tuple = tuple()

print(sys.getsizeof((test\_list)))  
print(sys.getsizeof((test\_tuple))) — кортеж будет весить меньше.

Вернёмся к изменяемым объектам.

Что делать, если мы захотим получить отдельный список в новую переменную?  
Нам поможет копирование.

Сделать это можно разными способами.  
L2 = L1[:] — популярный метод копирования, при котором мы, по сути, создаём новый объект из среза со всеми элементами старого списка.

В итоге L2 будет ссылаться уже на новый список, не связанный с оригинальным списком L1.

Такой приём сработает со списками, но не сработает, например, со словарями (у него и индексов нет для среза).

Поэтому более универсальный способ — использование метода copy().  
  
dist = {1: 2, 3: 4}  
x = dist.copy()  
x[1] = 5  
print(x, dist) — значение по ключу 1 будет разным в этих двух объектах.

Сам метод встроен в объекты, поэтому можно использовать его напрямую, но также существует и подключаемая стандартная библиотека copy.

Ссылки на официальную документацию и русский сайт с примерами работы с этой библиотеки:

* [copy — Shallow and deep copy operations (англ.)](https://docs.python.org/3/library/copy.html)
* [Модуль copy — поверхностное и глубокое копирование объектов](https://pythonworld.ru/moduli/modul-copy.html)

Использовать её можно так:  
import copy

dist = {1: 2, 3: 4}  
x = copy.copy(dist)  
x[1] = 5  
print(x, dist)

По сути, эти вызовы будут эквивалентны (разве что в первом варианте будет меньше символов). Но в модуле copy есть ещё одна полезная функция, которая называется deepcopy.

Чтобы понять всю её прелесть, надо сперва поговорить о сложных составных структурах данных.

Это те структуры, которые хранят в себе не просто числа, как в примерах выше, а другие структуры данных. Например, список списков:  
list\_of\_lists = [[1, 2, 3], 4, [5, 6, 7]]

или словарь списков:  
dict\_of\_lists = {1: [1, 2, 3], 2: [4, 5, 6]}

(Кстати, попробуйте использовать список в качестве ключа у словаря — вылезет ошибка, которая скажет вам о том, что изменяемые типы данных не могут быть использованы в качестве ключей.)

В целом, структура может быть совершенно любой комбинацией, так как каждый вложенный список/словарь и тому подобное может хранить в себе другие вложенные списки/словари и тому подобное.

Эта особенность настолько удобна и популярна, что, по сути, выросла в отдельный формат данных — JSON. Этот формат очень распространён и используется практически со всеми языками программирования, а суть его проста: в словаре хранятся ключи и значения, значениями могут быть другие словари/списки/строки/числа…

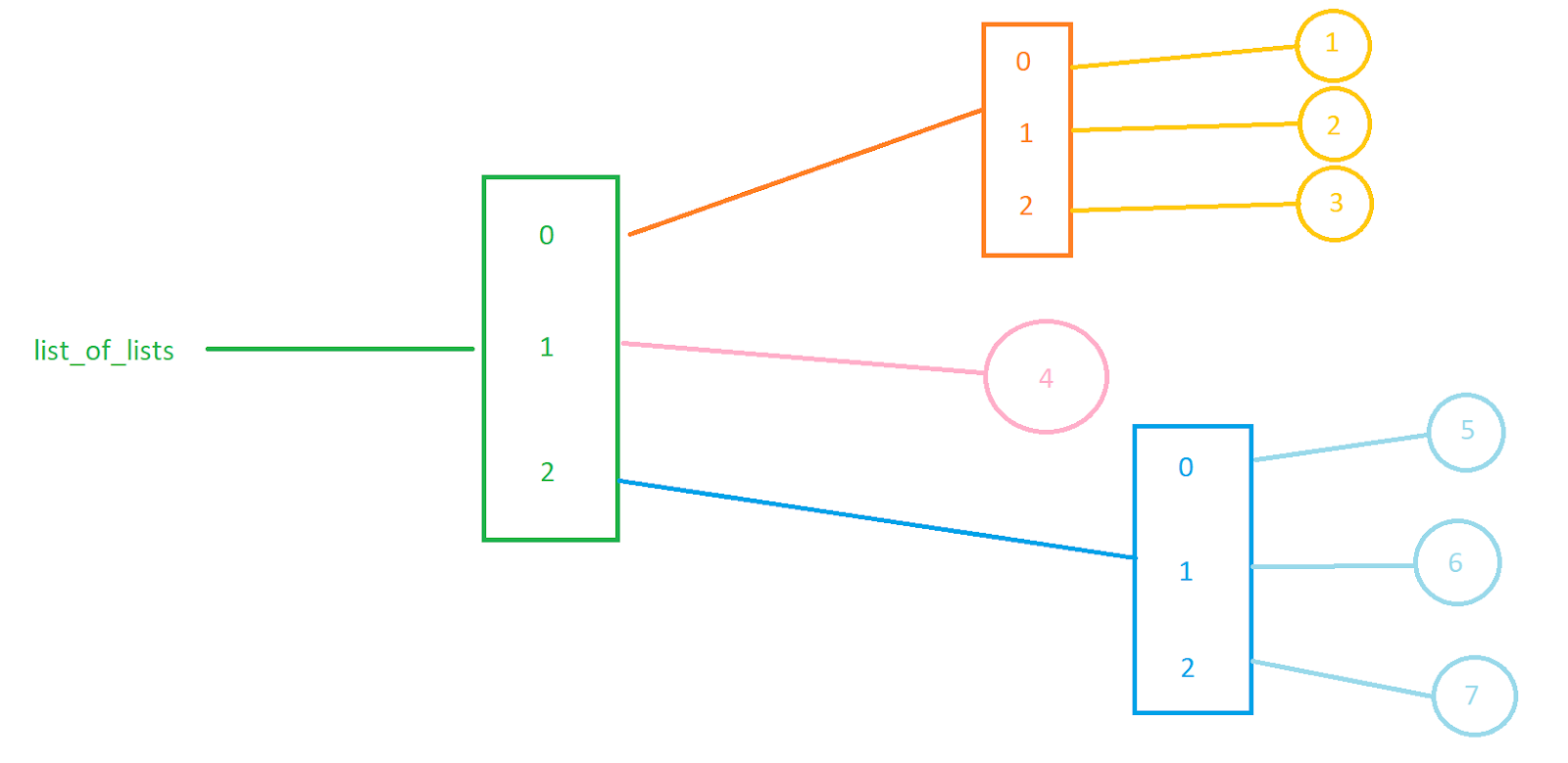
И это позволяет собирать информацию хоть и в сложные, но понятные и хорошо структурированные блоки данных.

**Работа с вложенными объектами**

Но при работе с подобным мощным инструментом стоит помнить и о его особенностях.

Важно понять, какую роль играют ссылки в структуре такого объекта.

list\_of\_lists = [[1, 2, 3],4, [5, 6, 7]]



Автор Максим Л.В

list\_of\_lists будет ссылаться на оригинал, у которого внутри есть ссылки на три любых объекта.

Первая ссылка будет ссылаться на другой список, у которого внутри тоже лежат три ссылки, которые, в свою очередь, ведут к числам 1, 2 и 3.

Вторая ссылка ведёт к числу 4.

Третья ссылка ведёт к списку с тремя ссылками внутри, а они ведут к числам 5, 6, 7.

Обращение к элементам.  
По сути, нам надо выстраивать цепочку из ссылок:  
list\_of\_lists[0] — ссылка на первую ссылку внутри списка.  
list\_of\_lists[0][0] — ссылка на число внутри списка, который лежит внутри оригинального списка.

Если мы напишем  
list\_of\_lists[0][0] = 10,  
то эти изменения будут отражены в каждом объекте, который ссылался на список внутри list\_of\_lists.

С текущим примером всё легко, но это также значит, что вложенный список может быть и в других списках.

Например, мы можем вынести один простой список в переменную:

x = [1, 2, 3].

list\_of\_lists = [x, 4, [5, 6, 7]] — а дальше добавлять его в другие списки  
через переменную.

list\_of\_lists\_2 = [x, [8], [9, 10, 11]]

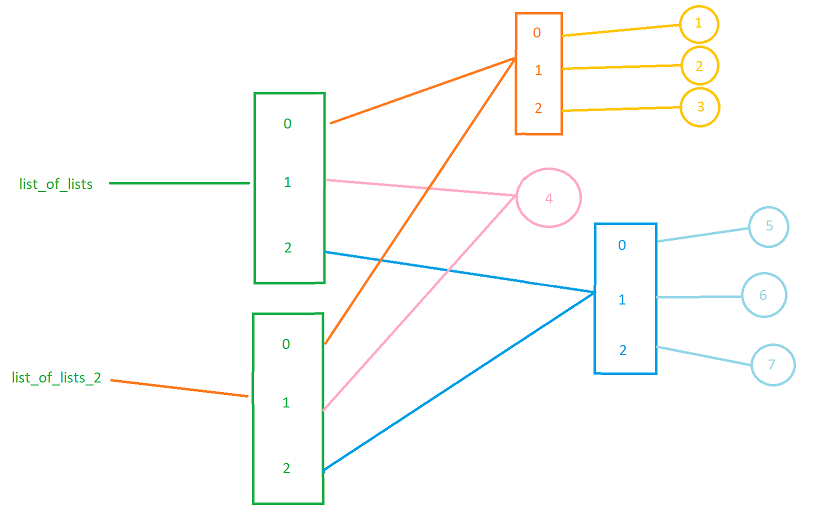
list\_of\_lists[0][0] = 10 — изменив элемент в этом списке:

print(x, list\_of\_lists\_2, list\_of\_lists, sep='\n'), мы увидим, что это изменение коснулось и x, и list\_of\_lists\_2.

Похожая особенность касается и копирования.  
Сперва давайте определимся с тем, что такое копирование — copy().  
По сути, это создание объекта с теми же вложенными объектами.

В Python copy() буквально копирует объект, и, если в объекте есть ссылки, они тоже копируются.

list\_of\_lists = [[1, 2, 3], 4, [5, 6, 7]]  
list\_of\_lists\_2 = list\_of\_lists.copy()



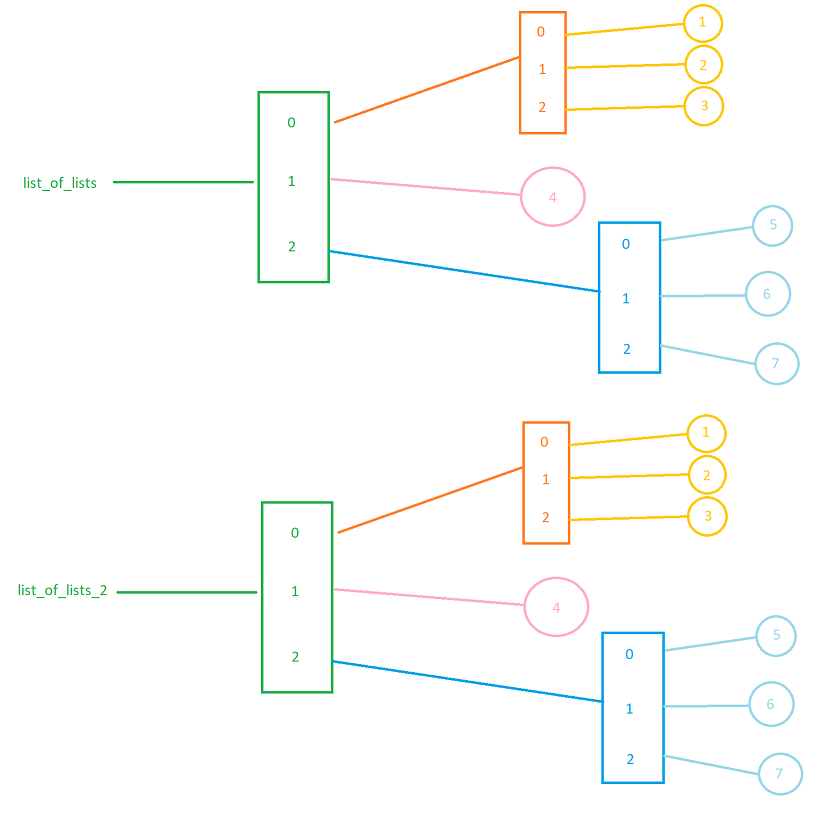
Автор Максим Л.В

Возможно, вы уже увидели проблему, которая кроется в таком подходе.  
list\_of\_lists[0][0] = 10 — изменение во вложенном списке.  
print(list\_of\_lists)  
print(list\_of\_lists\_2)  
[[10, 2, 3], 4, [5, 6, 7]]  
[[10, 2, 3], 4, [5, 6, 7]] — будут отражаться и в копии списка, так как копия имеет точно такие же ссылки, какие были в оригинале.

В итоге мы как будто получаем не полную копию, а только поверхностную копию списка.

Именно эту проблему решает функция deepcopy из модуля copy.

list\_of\_lists = [[1, 2, 3], 4, [5, 6, 7]]  
list\_of\_lists\_2 = copy.deepcopy(list\_of\_lists)



Автор Максим Л.В

list\_of\_lists[0][0] = 10  
print(list\_of\_lists)  
print(list\_of\_lists\_2)  
[[10, 2, 3], 4, [5, 6, 7]]  
[[1, 2, 3], 4, [5, 6, 7]] — теперь изменения не отражаются в копии.

Deepcopy, по сути, рекурсивно прошла по всем вложенным спискам и к каждому применила copy(), создав копии всех вложенных объектов.

**Резюме**

Основные моменты

1. Переменные — это ссылки на область памяти с объектом, а не сам объект.
2. На один объект может ссылаться несколько переменных.
3. Если мы вносим какие-то правки в изменяемый объект, то это будет отражено во всех переменных, которые ссылались на этот объект.
4. Чтобы изменения в одной переменной не касались другой переменной, нужно использовать копирование объектов (чтобы каждая переменная ссылалась на свой объект).
5. Копирование можно выполнять при помощи срезов (сделать срез со всеми элементами) или при помощи специального метода copy (или deepcopy из модуля copy).