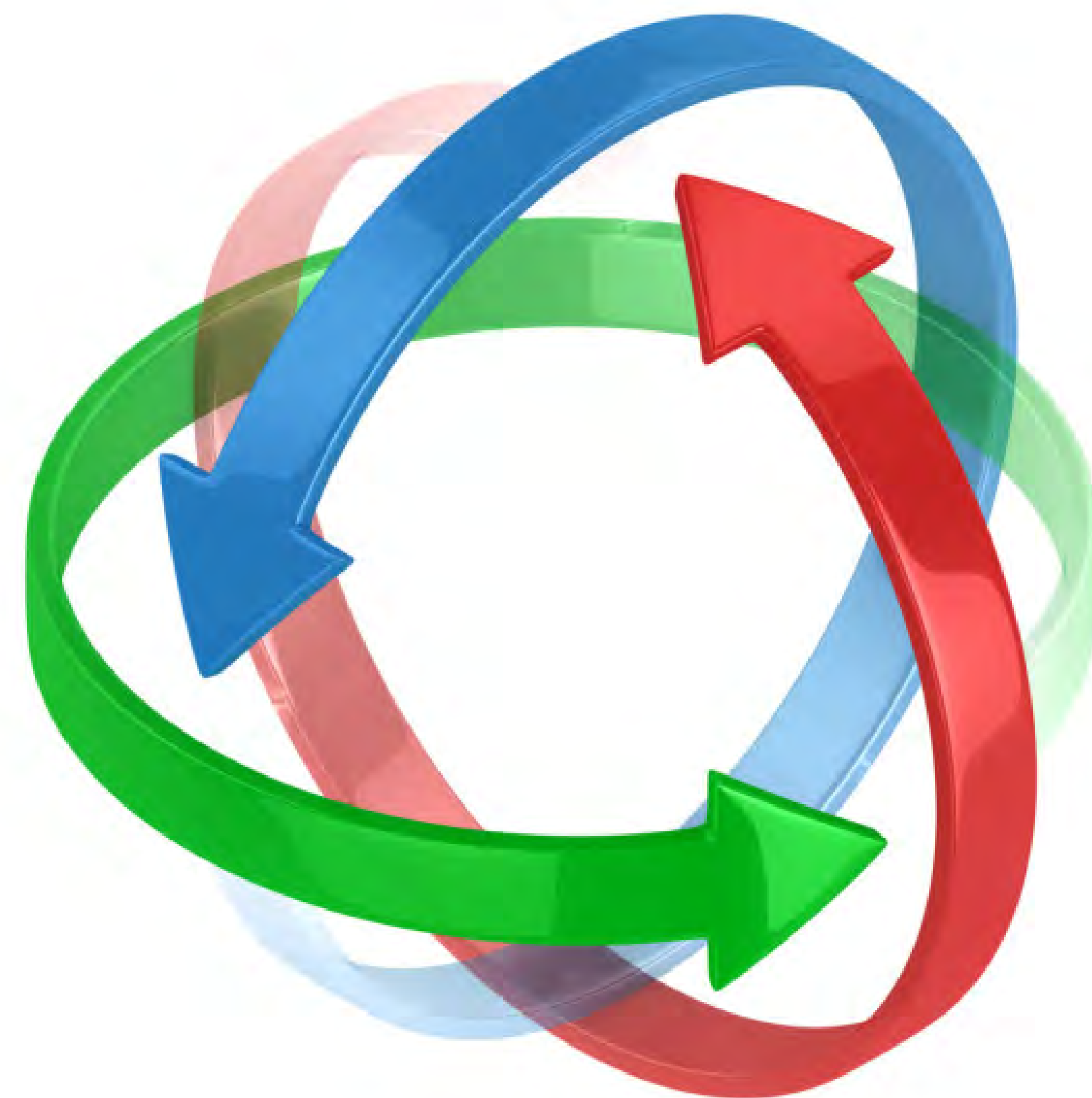


Python

5 урок. цикл While

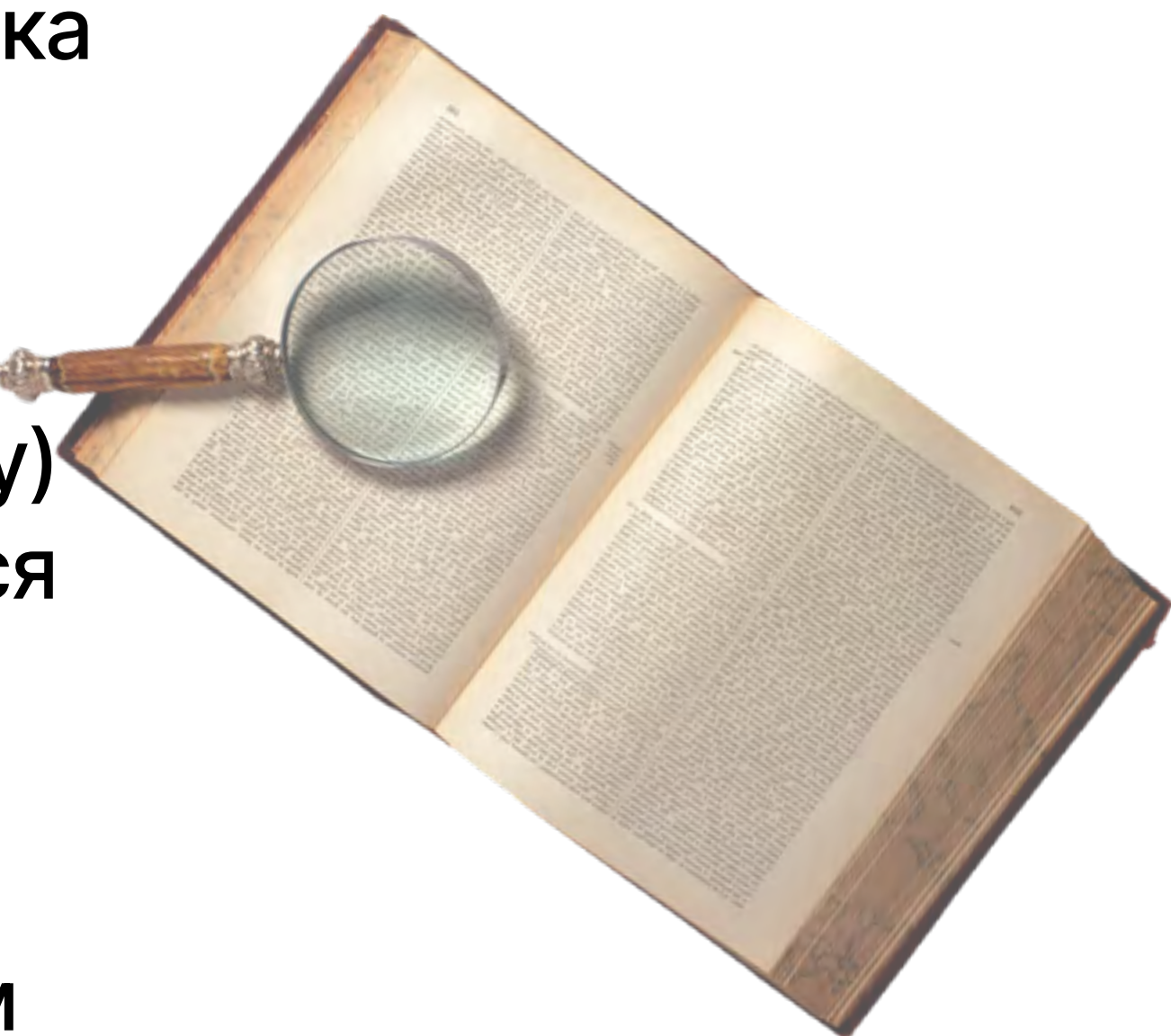
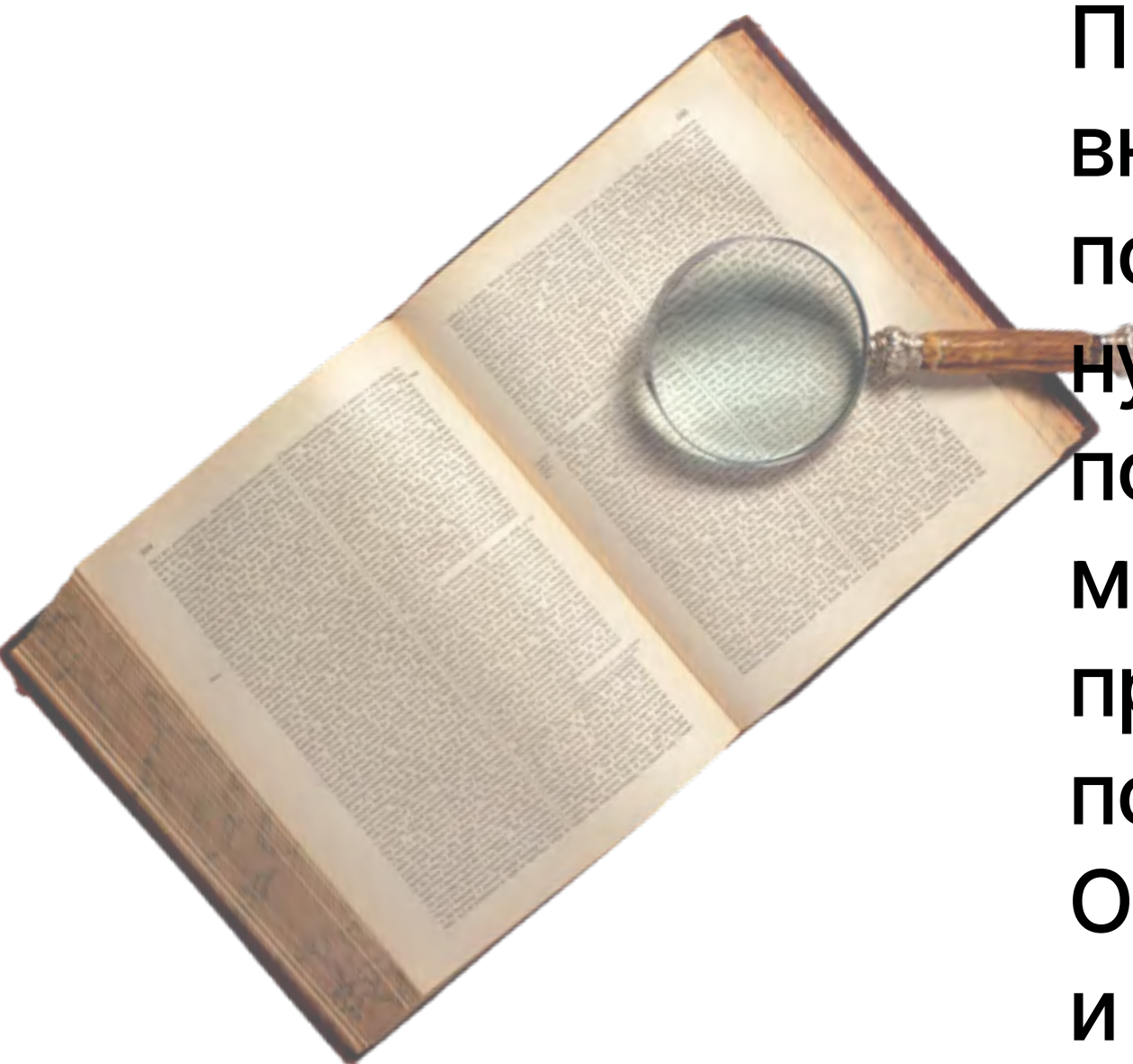
Циклы в Python

Программы, которые мы пишем, становятся всё сложнее и объемнее. Они все ещё очень далеки от реальных программ, где количество строк кода измеряется десятками и сотнями тысяч (а иногда и миллионами), но текущая сложность уже способна заставить напрячься людей без опыта. Начиная с этого урока, мы переходим к одной из самых сложных базовых тем в программировании – циклам.



Циклы в Python

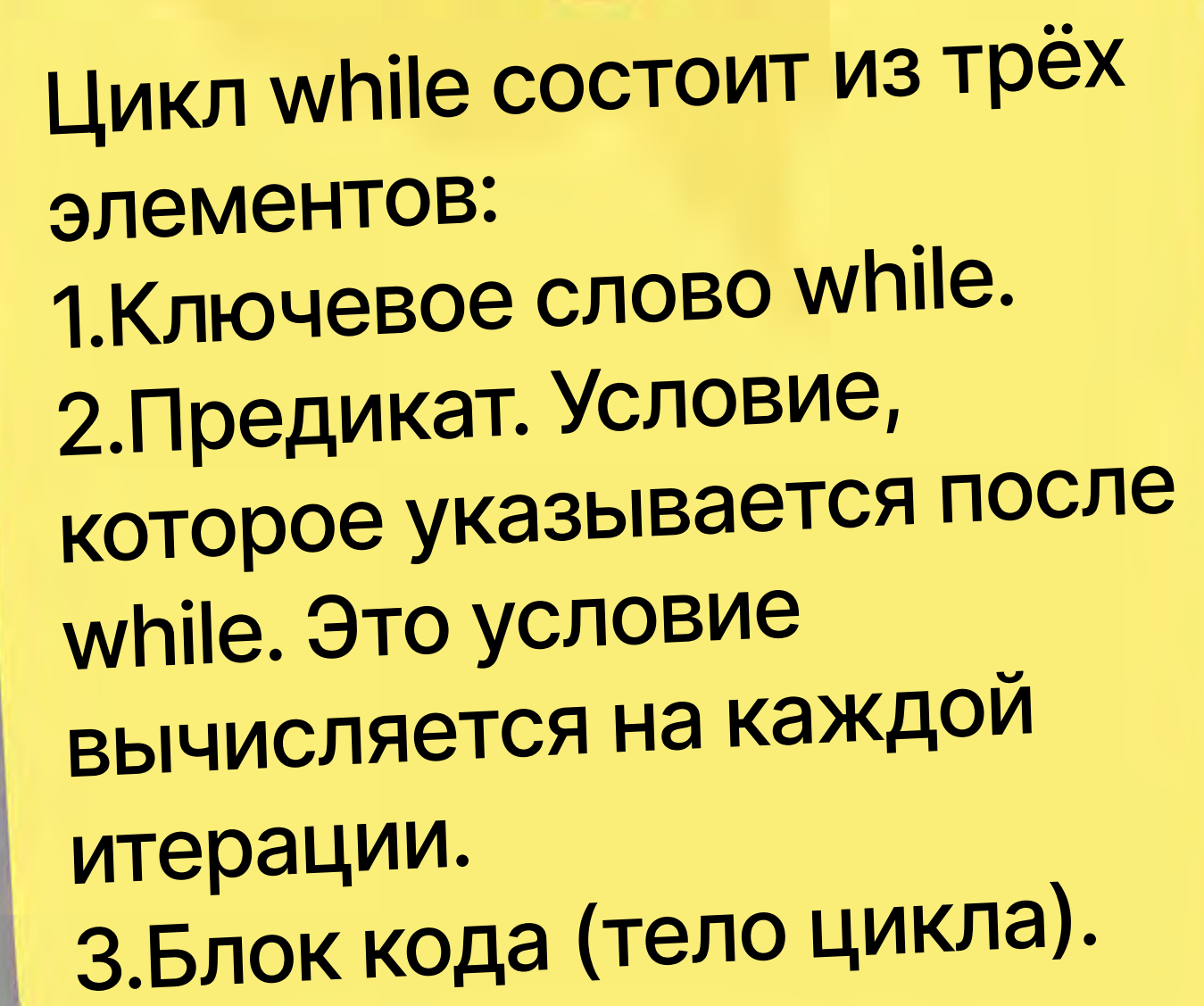
Представьте себе, что у нас есть книга и мы хотим найти внутри неё какую-то конкретную фразу. Саму фразу мы помним, но не знаем, на какой она странице. Как найти нужную страницу? Самый простой (и долгий) способ — последовательно просматривать страницы до тех пор, пока мы не найдем нужную. В худшем случае придется просмотреть все страницы, но результат мы всё равно получим. Именно этот процесс и называется алгоритмом. Он включает в себя логические проверки (нашли ли фразу) и перебор страниц. Количество страниц, которое придется посмотреть, заранее не известно, но сам процесс просмотра повторяется из раза в раз совершенно одинаковым образом. Для выполнения повторяющихся действий как раз и нужны циклы. Каждый повтор, в таком случае, называется итерацией.



Цикл While

Напишем код с простым циклом, который будет n раз выводить на экран строку 'Decode!':

```
n = int(input())  
counter = 0  
while counter < n:  
    print('Decode!')  
    counter = counter + 1
```



Цикл while состоит из трёх элементов:

1. Ключевое слово while.
2. Предикат. Условие, которое указывается после while. Это условие вычисляется на каждой итерации.
3. Блок кода (тело цикла).

Цикл While

Самое главное в цикле — завершение (выход). Процесс, который порождает цикл, должен в конце концов остановиться. Ответственность за остановку полностью лежит на программисте.

Обычно задача сводится к введению переменной, называемой «счётчиком цикла». Сначала он инициализируется, то есть ему задаётся начальное значение. В нашем примере это строка `counter = 0`. Затем в условии цикла проверяется, не достиг ли счётчик своего предельного значения.



В нашем примере предельное значение определяется переменной `n`. Если условие цикла не выполнено, то тело не выполняется и интерпретатор двигается дальше, выполняя инструкции после цикла. Но если условие цикла истинно, то выполняется тело, в котором находится ключевой элемент остановки — изменение счетчика. Обычно его делают в конце тела, и это изменение — одно из редких мест, где невозможно обойтись без переменной. В нашем примере за изменение отвечает строка:

```
counter = counter + 1.
```

Синтаксический сахар

Подобные конструкции `index = index + 1` в Python используются довольно часто, поэтому создатели языка добавили сокращённый вариант записи: `index += 1`.

Важно понимать, что отличия исключительно в способе записи.

Интерпретатор превращает сокращённую конструкцию в развёрнутую.

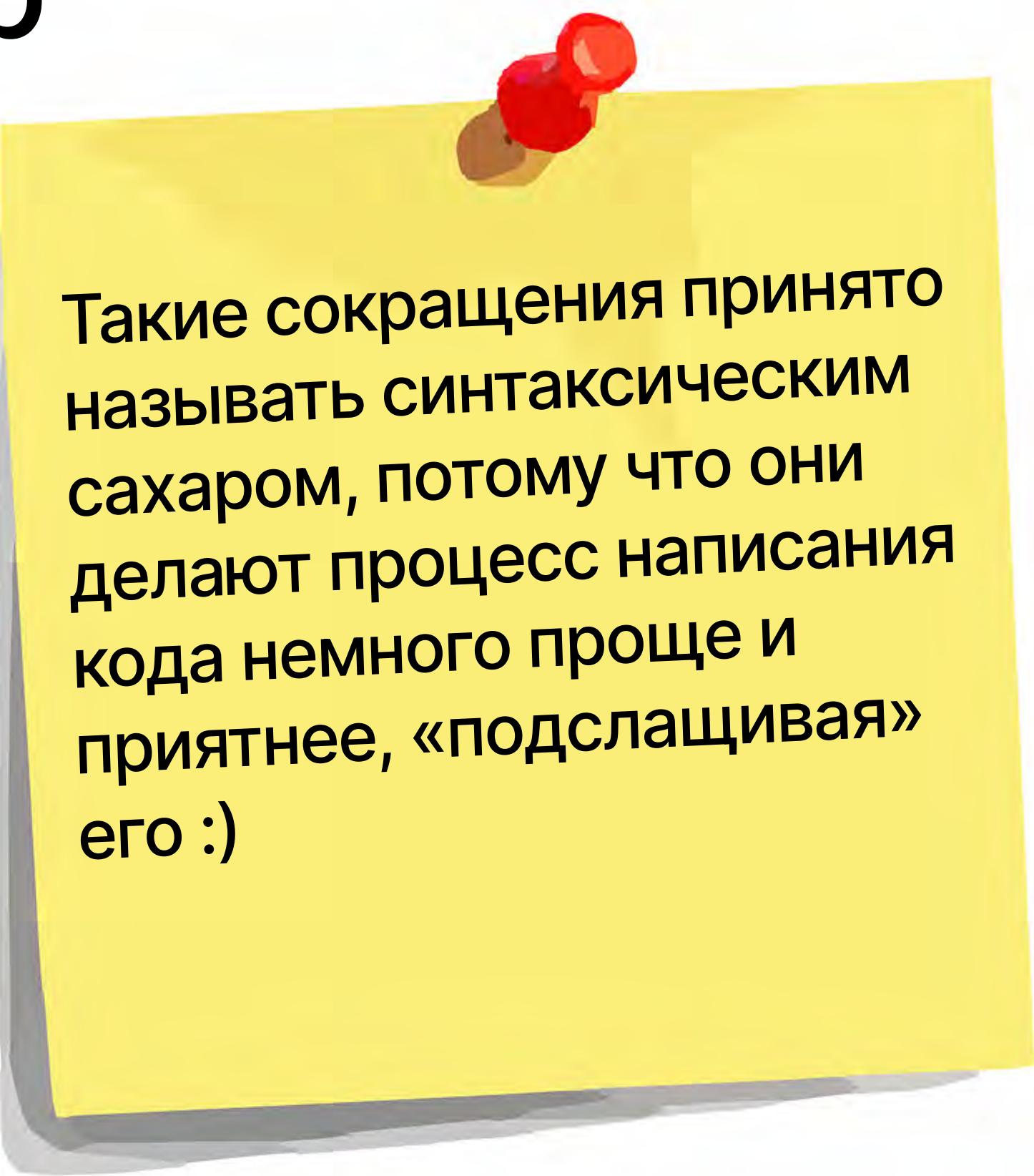
Существуют сокращённые формы для всех арифметических операций и для конкатенации строк:

`a = a + 1 → a += 1`

`a = a - 1 → a -= 1`

`a = a * 2 → a *= 2`

`a = a / 1 → a /= 1`



Такие сокращения принято называть синтаксическим сахаром, потому что они делают процесс написания кода немного проще и приятнее, «подслащивая» его :)



Бесконечный цикл

На этом моменте новички делают больше всего ошибок. Например, случайно забытое увеличение счётчика или неправильная проверка в предикате способны привести к заикливанию. Это ситуация, при которой цикл работает бесконечно и программа никогда не останавливается. В таком случае приходится её завершать принудительно (кто знает, может быть когда зависают реальные программы, в этот момент внутри них выполняется бесконечный цикл)

```
last_number = int(input())  
i = 1  
# Этот цикл никогда не остановится  
# и будет печатать всегда одно значение  
while i <= last_number:  
    print(i)  
print('finished!')
```



Бесконечный цикл

В некоторых случаях бесконечные циклы полезны. Как выглядит этот код:

```
while True:
```

```
    # Что-то делаем
```

Цикл не остановится до тех пор, пока не нажать Ctrl + C.

Однако в некоторых случаях бесконечный цикл делают намерено:


1. Если нужно производить какие-то действия с интервалом, и выходить из цикла лишь в том случае, когда внутри тела "зашито" условие выхода. Пример: функция, которая возвращает connection базы данных. Если связь с базой данных отсутствует, соединение будет пытаться (в цикле) установиться до тех пор, пока не установится.
2. Если вы пишете полноценный демон, который продолжительное время висит как процесс в системе и периодически производит какие-то действия. В таком случае остановкой цикла будет прерывание работы программы. Пример: скрипт, который раз в 10 минут "пингует" IP адреса и пишет в лог отчет о доступности этих адресов.

break и continue

Оператор break заставляет интерпретатор прервать выполнение цикла и перейти к следующей за ним инструкции:

```
counter = 0
while True:
    if counter == 10:
        break
    counter += 1
```

Цикл прервётся после того, как значение счетчика достигнет десяти.



******Эти операторы бывают весьма удобны, однако плохой практикой считается написание кода, который чересчур ими перегружен.

Существует похожий оператор под названием continue, однако он не прекращает выполнение всей конструкции, а прерывает лишь текущую итерацию, переходя затем в начало цикла: Классический пример вывода одних лишь чётных значений

```
z = 10
while z:
    z -= 1
    if z % 2 != 0:
        continue
    print(z, end=" ")
```