Лекция 4. Операторы выбора, циклов, переходов

# **Введение**

В мире программирования важным аспектом разработки эффективного и функционального кода является понимание и умелое использование различных управляющих конструкций. Среди них особое место занимают операторы выбора, операторы цикла и операторы перехода. Эти операторы являются основными инструментами, которые позволяют разработчикам управлять потоком выполнения программы, делать её более гибкой и адаптируемой к различным условиям.

## **Операторы выбора**

Операторы выбора, такие как if, else, elif в Python или switch в языках C-подобных, позволяют программе принимать решения и выполнять определенные действия в зависимости от условий. Их назначение заключается в том, чтобы предоставлять возможность выбора одного из нескольких возможных путей выполнения кода. Операторы выбора находят широкое применение во всех областях программирования, от простых скриптов до сложных алгоритмов и систем управления.

## **Операторы цикла**

Операторы цикла, такие как for, while и do-while, обеспечивают повторное выполнение блока кода до тех пор, пока выполняется определенное условие. Их основное назначение – автоматизация повторяющихся задач, что значительно упрощает работу с большими объемами данных и улучшает производительность программы. В различных приложениях операторы цикла используются для итерации по коллекциям данных, выполнения заданий в фоновом режиме и многого другого.

## **Операторы перехода**

Операторы перехода, такие как break, continue и goto, позволяют изменять стандартный поток выполнения программы. Они используются для выхода из циклов, пропуска итераций или перехода к определенным частям кода. Хотя операторы перехода, особенно goto, могут усложнять понимание программы, их грамотное использование позволяет создавать более эффективные и управляемые коды.

# **Операторы выбора (условия)**



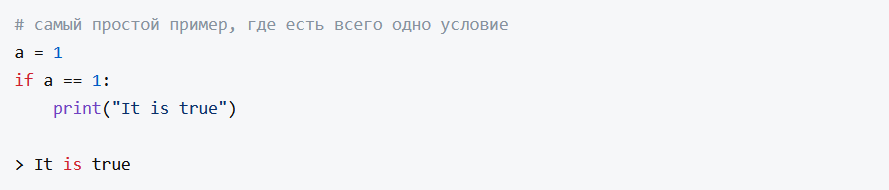
Фундаментальная важность условий для любого из языков программирования заключается в их возможности описывать большую часть логики работы программы.

if else — это оператор, управляющий условным ветвлением. Простыми словами это конструкция в Python, указывающая интерпретатору, следует ли выполнять определенный участок кода или нет.

Как и все прочие составные инструкции языка, оператор выбора также поддерживает свойство вложенности. Это означает, что использование if else позволяет создавать внутри программного модуля так называемое логическое ветвление.

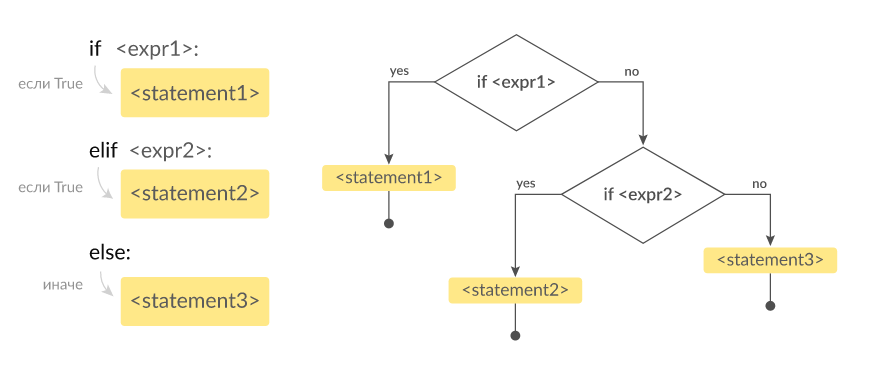
**Синтаксис**

Оператор if else в языке Python — это типичная условная конструкция, которую можно встретить и в большинстве других языков программирования.

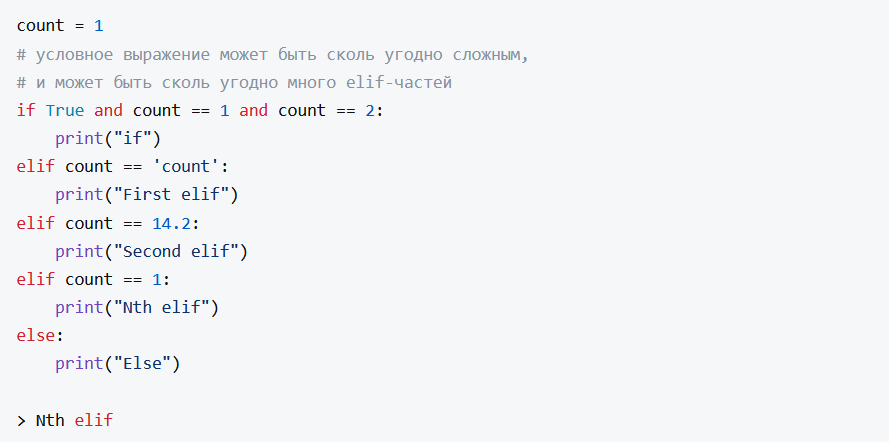


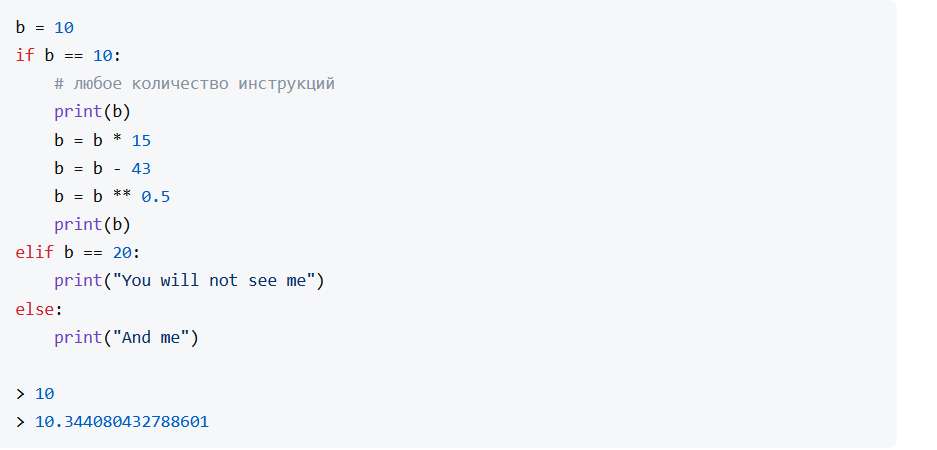
Синтаксически конструкция выглядит следующим образом:

* сначала записывается часть if с условным выражением, которое возвращает истину или ложь;
* затем может следовать одна или несколько необязательных частей elif (в других языках вы могли встречать else if);
* Завершается же запись этого составного оператора также необязательной частью else.



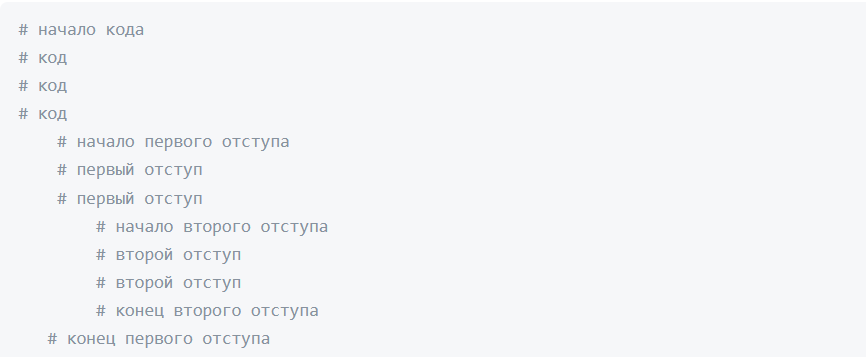
Принцип работы оператора выбора в Python

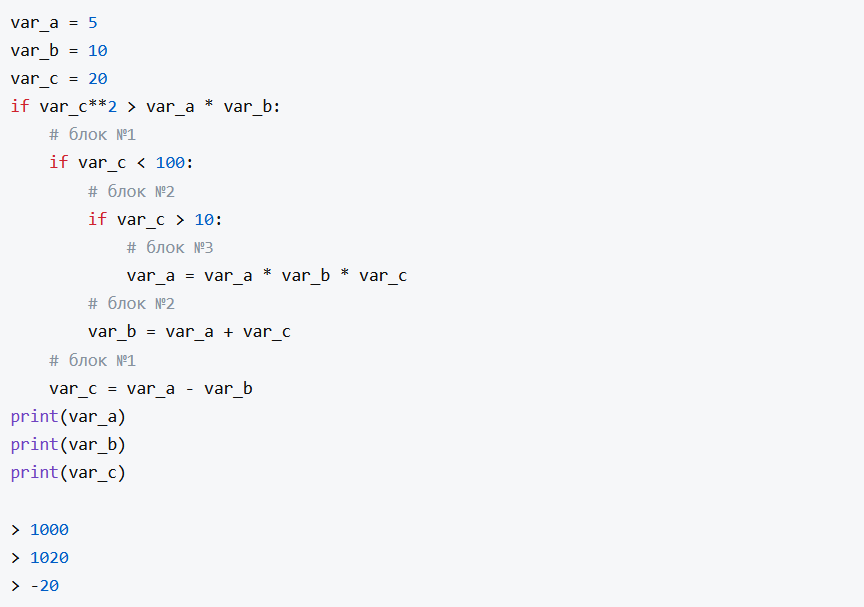
Для каждой из частей существует ассоциированный с ней блок инструкций, которые выполняются в случае истинности соответствующего им условного выражения.

То есть интерпретатор начинает последовательное выполнение программы, доходит до if и вычисляет значение сопутствующего условного выражения. Если условие истинно, то выполняется связанный с if набор инструкций. После этого управление передается следующему участку кода, а все последующие части elif и часть else (если они присутствуют) опускаются.

## **Отступы**

Отступы — важная и показательная часть языка Python. Их смысл интуитивно понятен, а определить их можно, как размер или ширину пустого пространства слева от начала программного кода.

Благодаря отступам, python-интерпретатор определяет границы блоков. Все последовательно записанные инструкции, чье смещение вправо одинаково, принадлежат к одному и тому же блоку кода. Конец блока совпадает либо с концом всего файла, либо соответствует такой инструкции, которая предшествует следующей строке кода с меньшим отступом.

Таким образом, с помощью отступов появляется возможность создавать блоки на различной глубине вложенности, следуя простому принципу: чем глубже блок, тем шире отступ.

## **Примеры**

Рассмотрим несколько практических примеров использования условного оператора.

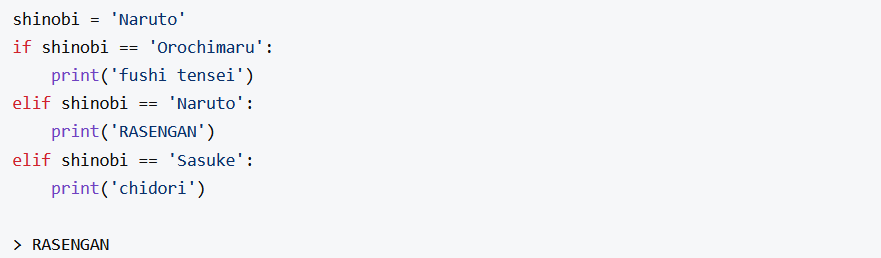
Пример №1: создание ежедневного бэкапа (например базы данных):

Пример №2: Проверка доступа пользователя к системе. В данном примере if проверяет наличие элемента в списке:

Пример №3: Валидация входных данных. В примере к нам приходят данные в формате json. Нам необходимо выбрать все записи определенного формата:

## **Оператор elif**

elif позволяет программе выбирать из нескольких вариантов. Это удобно, например, в том случае, если одну переменную необходимо многократно сравнить с разными величинами.

Такая конструкция может содержать сколь угодно большую последовательность условий, которые интерпретатор будет по порядку проверять.

Но помните, что первое условие всегда задается с if

Также не стоит забывать, что как только очередное условие в операторе оказывается истинным, программа выполняет соответствующий блок инструкций, а после переходит к следующему выражению.

Из этого вытекает, что даже если несколько условий истинны, то исполнению подлежит все равно максимум один, первый по порядку, блок кода с истинным условием.

Если ни одно из условий для частей if и elif не выполняется, то срабатывает заключительный блок под оператором еlse (если он существует).

## **Заглушка pass**

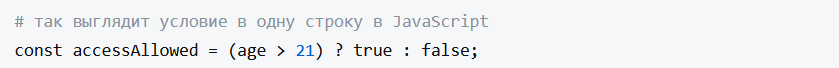
Оператор-заглушка pass заменяет собой отсутствие какой-либо операции.

Он может быть весьма полезен в случае, когда в ветвлении встречается много elif-частей, и для определенных условий не требуется выполнять никакой обработки.

Наличие тела инструкции в Python обязательно

## **Тернарная форма записи оператора условия**

Во многих языках программирования условие может быть записано в одну строку. Например, в JavaScript используется тернарный оператор:

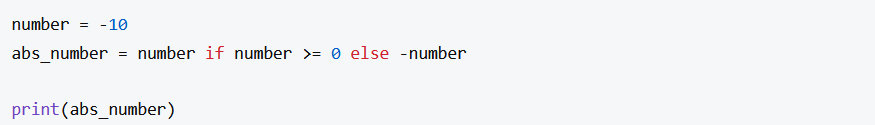
Читается это выражение так: если age больше 21, accessAllowed равен true, иначе — accessAllowed равен false.

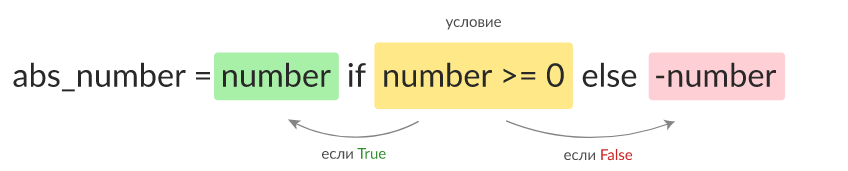
В Python отсутствует тернарный оператор

Вместо тернарного оператора, в Питоне используют инструкцию if else, записанную в виде выражения (в одно строку):



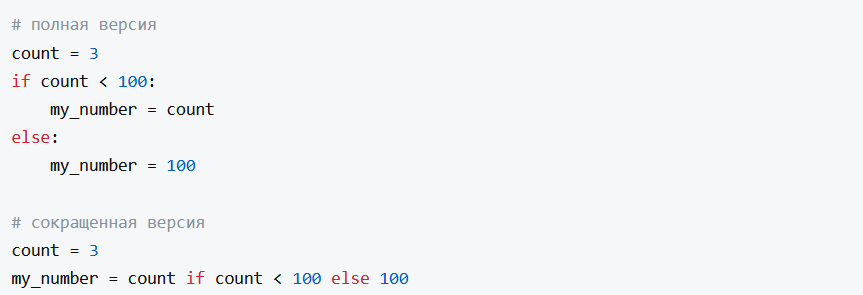
Пример:

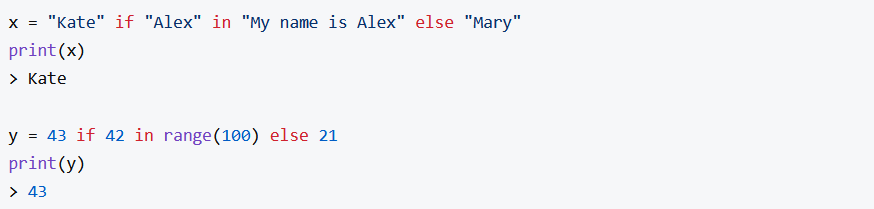
Такая конструкция может показаться сложной, поэтому для простоты восприятия, нужно поделить ее на 3 блока:



Для простоты восприятия if-else, записанного одной строкой, разделите выражение на 3 блока

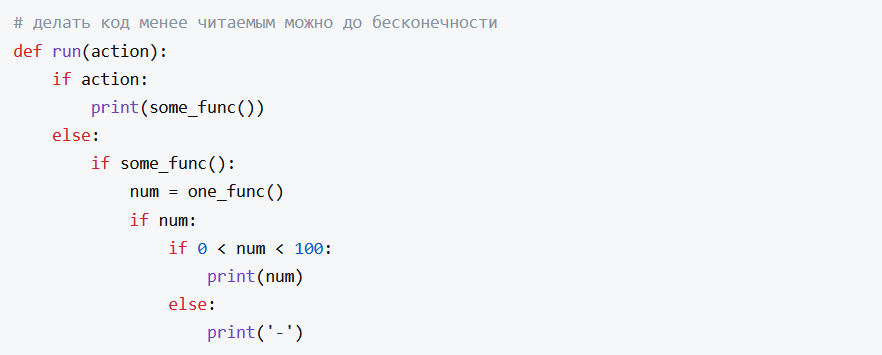
Стоит ли использовать такой синтаксис? Если пример простой, то однозначно да:

Вполне читаемо смотрятся и следующие 2 примера:

Но если вы используете несколько условий, сокращенная конструкция усложняется и становится менее читаемой:

## **Вложенные условия**

Ограничений для уровней вложенности в Python не предусмотрено, а регулируются они все теми же отступами:

Стоит ли использовать такие вложенности? Скорее нет, чем да. Одно из положений [Python Zen](https://pythonchik.ru/osnovy/dzen-python-pep20) гласит:

*Flat is better than nested (развернутое лучше вложенного).*

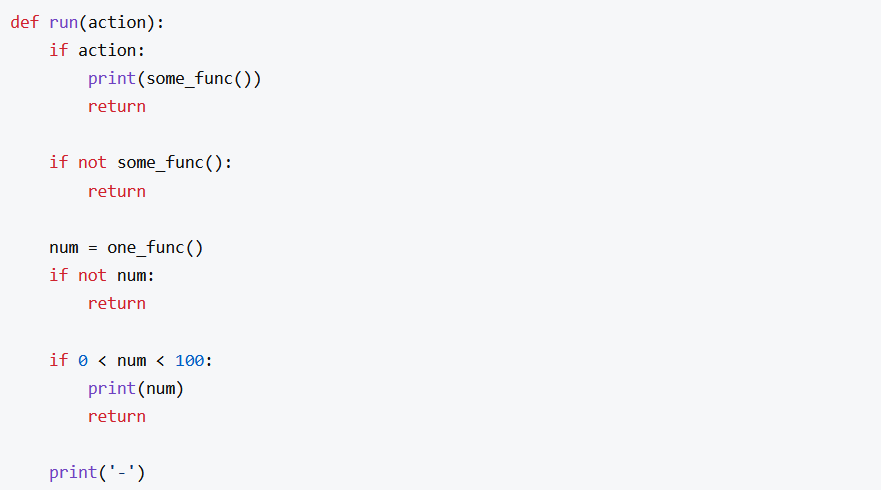
Большая вложенность имеет следующие недостатки:

* становится трудно легко найти, где заканчивается конкретный блок;
* код становится менее читаемым и сложным для понимания;
* возможно, придется прокручивать окно редактора по горизонтали.

Но что делать, если в скрипте не получается уйти от большой вложенности if-else? 🤷‍♂️

Чтобы уйти от большой вложенности, попробуйте не использовать оператор else.

Пример выше, можно записать следующим образом:

**Конструкция switch case**

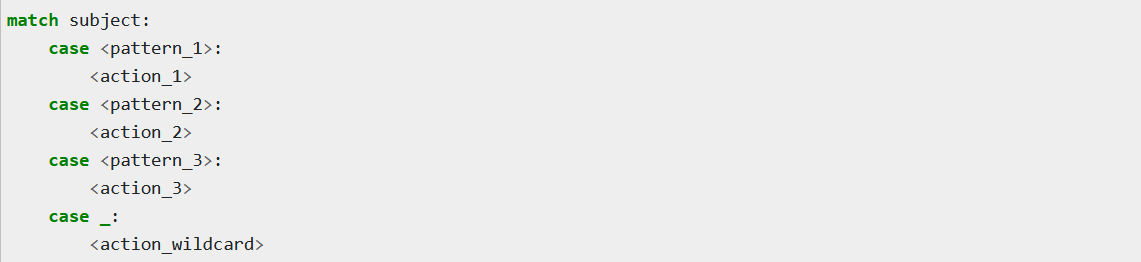
В python данная конструкция отсутствует, однако начиная с версии 3.10 была введена аналогичная конструкция – match/case.

новая конструкция match/case называется Structural pattern matching (соответствие структуре шаблона). Оператор match был введен для того, чтобы быть больше чем просто похожим на оператор switch, который присутствует во многих других языках программирования. PEP 634, 635 и 636 содержат много информации о том, что приносит конструкция match/case в Python 3.10, а также обоснование его добавления.

Во многих случаях конструкция match/case, может упростить и повысить читабельность кода Python. Этот материал акцентирует внимание на практическом использовании конструкции match/case, чтобы писать короткий и красивый код.

Шаблоны в операторах case, конструкции match/case состоят из последовательностей, словарей, примитивных типов данных (int, float,str и т.д.), а также экземпляров классов. Сопоставление с образцом позволяет программам извлекать информацию из сложных типов данных, переходить к структуре данных и применять определенные действия на основе различных форм данных.

Общий синтаксис конструкции match/case:

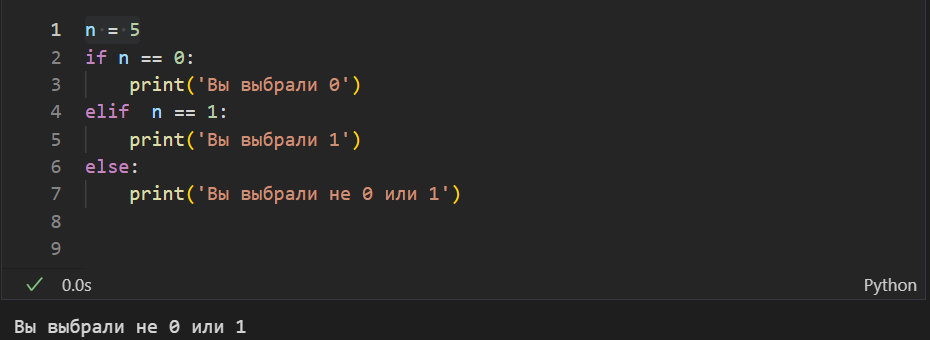


Оператор match принимает выражение subject и сравнивает его значение с последовательными шаблонами, заданными как один или несколько блоков case. В частности, сопоставление с образцом работает следующим образом:

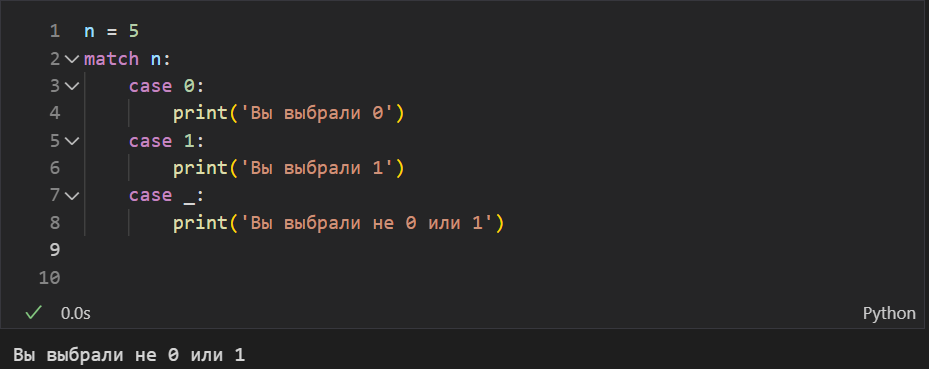
* использование данных с типом и формой (subject);
* оценка subject в заявлении match;
* сравнение subject с каждым шаблоном в заявлении case сверху вниз, пока совпадение не будет подтверждено.
* выполнение действия action, связанного с шаблоном подтвержденного совпадения;
* если точное совпадение не подтверждено, то в качестве совпадающего случая будет использоваться последний case c подстановочным знаком '\_', если он указан. Если точное совпадение не подтверждено и case \_: - не существует, то весь блок match не выполняется.

Обратите внимание, что большинство литералов сравниваются по равенству. Однако синглтоны: True, False и None сравниваются по идентичности.

Базовые приемы использования конструкции match/case:



Теперь вместо конструкции if‑else будем использовать новую конструкцию match/case:



На простых примерах разница в скорости выполнения незаметна, однако в более сложных задачах конструкция match/case имеет некоторое преимущество во времени выполнения.

# **Операторы цикла**

Циклы в Python – инструменты, упрощающие работу с кодом. В программировании все циклично и последовательно. Написание любой программы состоит из прохождения определенных этапов, задачи в которых могут повторяться. Вот чтобы не тратить время на повторяющиеся элементы и были придуманы циклы. К слову, они есть в любом языке программирования, не только в Python.

В Python основных циклов всего два – «while» и «for». Первый используется тогда, когда заранее известно количество итераций, а второй – когда нужно выполнить перебор элементов. Но мало знать принципы работы циклов, в работе необходимы еще и операторы, которые делают «while» и «for» бесконечными или вложенными, прерывают их или меняют. Ниже в статье постарались объяснить все это доступным языком с примерами.

Цикличные задачи имеют место даже в повседневной жизни. Например, различные списки продуктов, целей на день, назначенных встреч и т. д. Приходя в супермаркет, человек ориентируется на список, покупая все товары, которые в нем указаны. В сфере программирования циклы дают возможность осуществлять повтор тех или иных действий в зависимости от соблюдения имеющегося условия. Тем самым обеспечивается выполнение многократной последовательности инструкции.

В рамках данной темы можно выделить ряд базовых понятий:

* Единоразовое выполнение — так называемая итерация.
* Тело цикла — последовательность кода, которую можно исполнить многократно.

Также вы можете создавать вложенные циклы. В этом случае программа произведет запуск внешнего цикла и в первой его итерации перейдет во вложенный. После этого она вновь вернётся к началу внешнего и опять запустит внутренний. Эти действия будут выполняться вплоть до момента, когда последовательность не завершится или не прервется. Циклы очень полезны для выполнения задач, связанных с перебором определенного количества элементов в заданном списке.

Циклы в Python позволяют сделать из сложных задач более простые. С помощью них вы можете настроить поток программы так, чтобы код повторялся определенное количество раз. Иными словами, вам не придется постоянно прописывать один и тот же код. Допустим, что перед вами стоит задача вывода первых 20 натуральных чисел. Вы, конечно, можете применить оператор «print» 20 раз, но гораздо более удобный вариант — вывести их внутри цикла, который выполняет до 20 итераций.

Применительно к Python можно выделить следующие плюсы использования циклов:

* Возможность перебора элементов структур данных (массивов или связанных списков).
* Возможность многократного использования кода.
* Отсутствие необходимости прописывать один и то же код несколько раз.

|  |  |
| --- | --- |
| Оператор цикла | Описание |
| «For» | Данный цикл применяется в том случае, если нужно выполнять тот или иной фрагмент вплоть до момента, когда будет исполнено заданное условие. Цикл «for», который иногда называют циклом с предусловием, рекомендуется применять в тех ситуациях, когда вы заранее знаете численность итераций. |
| «While» | Этот цикл применяется в тех случаях, когда численность итераций заранее не известна. Блок операторов выполняется до тех пор, пока не будет выполнено условие, указанное в цикле. У него есть альтернативное название — цикл с предварительной проверкой условия. |
| «do-while» | Он продолжается вплоть до момента, когда будет выполнено заданное условие. Альтернативное название — цикл с постусловием в Python. Применяется в ситуациях, при которых стоит задача как минимум однократного выполнения цикла. |

## **Примеры циклов в Python**

**Цикл «for»**

Данный цикл в языке Python осуществляет поворотное выполнение сформированного кода на основе переменной или счетчика. Он применяется, только если нужно перебрать элементы заранее известное количество раз. Допустим, что у вас есть список. Сначала из него будет взят первый элемент, затем второй и т. д. При этом с каждым из этих элементов будет выполняться действие, заранее предписанное в теле «for». Приведем пример цикла в Python:

Функция «range()» позволяет упростить работу. Она демонстрирует нужное число повторов последовательности. Тем самым данная функция цикла для Python конкретизирует элементы из списка «for», которые вам нужны в данный момент. В скобках вы увидите от 1 до 3 чисел:

* 1 число говорит, что следует осуществить проверку всех чисел от 0 и до него;
* 2 числа свидетельствуют о необходимости проверить все числа, которые располагаются между первым и вторым;
* 3 числа создадут список от первого до второго с шагом, который будет равен третьему.

**Цикл «while»**

Англоязычное слово «while» можно перевести как «до тех пор, как». Данный цикл является довольно универсальным. Он может слегка напоминать условие «if». При этом код «while» выполняется неоднократно. Написание его условия осуществляется до тела цикла. Вслед за первым выполнением программа перемещается обратно к заголовку и опять повторяет те же операции.

В конечном итоге последовательность завершается в том случае, если условие цикла уже не может быть выполнено. Проще говоря, когда оно перестаёт быть истинным. Отличие этого цикла от «for» заключается в том, что разработчик не имеет информации о численности проверок.

## **Выход из цикла Python или его смена**

Чтобы произвести выход из цикла в программировании на Python применяется оператор «break». Он осуществляет досрочное завершение, обходя «else». Для чего же это нужно? К примеру, при выполнении кода была выявлена ошибка, и, следовательно, последующая работа будет безрезультативной.

Ещё одна инструкция, вносящая изменения в цикл, — «continue». При написании данного оператора внутри кода программа игнорирует все остальные инструкции до конца цикла. Далее начинается следующая итерация.

## **Бесконечные и вложенные циклы в Python**

**Бесконечные циклы**

В этом случае не выполняется условие выхода. Скажем, цикл «while» является таковым в том случае, если его условие не может быть ложным. Он полезен, к примеру, для создания программы «Часы», беспрерывно демонстрирующей время.

В качестве примера рассмотрим следующий код:

Num = 4

while num < 6:

print «Добрый день»

Как вы можете заметить, переменная всегда будет являться числом 4, так как не прописано ее повышение. По этой причине на экране будет отображаться лишь «Добрый день».

Во многих случаях цикл не должен быть бесконечным, так как это является одним из факторов нестабильности при работе программы. Для его завершения следует использовать комбинацию клавиш: «CTRL + C». Однако есть программы, которые вообще не подразумевают возможность выхода. К примеру, операционные системы, прошивки микроконтроллеров и т. д.

**Вложенные циклы**

Речь пойдет о вложенных «for» и «while». Рассмотрим несколько вариантов применения таких циклов. Их на постоянной основе применяют в процессе обработки двухмерных списков.

Обратите внимание на пример создания двумерного списка и вывода его на экран посредством «print».

d = [[1,2,3],[4,5,6]]

for i in range(2):

for j in range(3):

print(d[i][j])

После использования одного «break» в Python произойдёт выход лишь из одного цикла. Ниже вы можете посмотреть на условие, которое позволяет выйти сразу из двух.

toExit = False

while True:

while True:

toExit = doSomething()

if toExit:

break

if toExit:

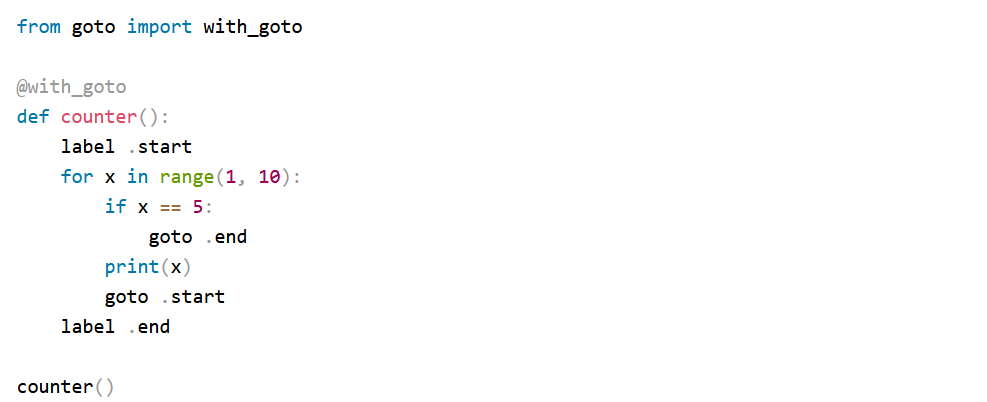
break

Это называется циклом в цикле. Причем они оба являются бесконечными. После того как функция «doSomething» вернет «True», выполнение кода прекратится. Затем друг за другом активируются «break» в каждом из циклов.

# **Операторы перехода**

Изучая Python, вы заметите, что здесь нет места для goto и меток. Python — это язык с чётко выстроенной структурой и высокой читабельностью, где не приемлем хаос. Рассмотрим альтернативы, предоставляемые Python.

Прямого аналога команде goto в Python нет. Но существуют приёмы, эмулирующие её поведение. Вот пример имитации goto с помощью сторонней библиотеки:



Хоть данный код и работает, рекомендуется придерживаться структурированного сценария управления потоком в Python вместо применения goto. Именно к этому стремится Python 🐍.

# **Заключение**

Понимание назначения, области применения и способов реализации операторов выбора, циклов и переходов является фундаментальным для любого программиста. Эти управляющие конструкции не только помогают писать более эффективный код, но и позволяют решать сложные задачи, управлять большими объемами данных и обеспечивать гибкость и адаптивность программного обеспечения. В этой статье мы рассмотрим каждый из этих операторов более подробно, проанализируем примеры их использования и обсудим лучшие практики их применения в реальных проектах.

# **Список литературы**

1. https://gb.ru/blog/tsikly-v-python/

2. https://docs-python.ru/tutorial/tsikly-upravlenie-vetvleniem-python/konstruktsija-match-case/

3. https://pythonworld.ru/osnovy/instrukciya-if-elif-else-proverka-istinnosti-trexmestnoe-vyrazhenie-ifelse.html

4. https://pythonchik.ru/osnovy/python-if-elif-else

5. https://informatics.msk.ru/mod/book/tool/print/index.php?id=46386

6. https://python-school.ru/blog/python/python-loop-operators/

7. https://sky.pro/wiki/python/pryzhki-po-kodu-v-python-est-li-v-nyom-goto-label/

8. https://pythonworld.ru/osnovy/cikly-for-i-while-operatory-break-i-continue-volshebnoe-slovo-else.html

9. https://itproger.com/course/python/6

10. https://pyneng.readthedocs.io/ru/latest/book/06\_control\_structures/  
break\_continue\_pass.html

11. https://ravesli.com/break-continue-python/

12. https://tproger.ru/links/python-goto