Ветвление. Условный оператор

Ход выполнения программы может быть *линейным*, то есть таким, когда выражения выполняются друг за другом, начиная с первого и заканчивая последним. Ни одна строка кода программы не пропускается.

Однако чаще в программах бывает не так. При выполнении кода, в зависимости от тех или иных условий, некоторые его участки могут быть опущены, в то время как другие – выполнены. Иными словами, в программе может присутствовать *ветвление*, которое реализуется **условным оператором – особой конструкцией языка программирования**.

Проведем аналогию с реальностью. Человек живет по расписанию. Можно сказать, расписание – это алгоритм для человека, его программный код, подлежащий выполнению. В расписании на 18.00 стоит поход в бассейн. Однако экземпляр биоробота класса Homo sapiens через свои рецепторы-сенсоры получает информацию, что воду из бассейна слили. Разумно было бы отменить занятие по плаванию, т. е. изменить ход выполнения программы-расписания. Одним из условий посещения бассейна должно быть его функционирование, иначе должны выполняться другие действия.

Подобная нелинейность действий может быть реализована в компьютерной программе. Например, часть кода будет выполняться лишь при определенном значении конкретной переменной. В языках программирования используется приблизительно такая конструкция условного оператора:

if логическое\_выражение {

выражение 1;

выражение 2;

…

}

Перевести на человеческий язык можно так: **если логическое выражение возвращает истину, то выполняются выражения внутри фигурных скобок**; если логическое выражение возвращает ложь, то код внутри фигурных скобок не выполняется. С английского "if" переводится как "если".

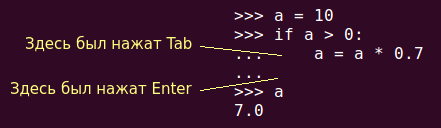
Конструкция if логическое\_выражение называется **заголовком условного оператора**. Выражения внутри фигурных скобок – **телом условного оператора**. Тело может содержать как множество выражений, так и всего одно или даже быть пустым.

Пример использования условного оператора в языке программирования Python:

**if** n < 100:

b = n + a

В Питоне вместо фигурных скобок используется двоеточие. Обособление вложенного кода, т. е. тела оператора, достигается за счет отступов. В программировании принято делать отступ равным четырем пробелам. Однако также можно использовать клавишу табуляции (Tab) на клавиатуре. Большинство сред программирования автоматически делают отступ, как только вы поставите двоеточие и перейдете на новую строку. Однако при работе в интерактивном режиме надо делать отступы вручную.



Нахождение в теле условного оператора здесь обозначается тремя точками. При создании файла со скриптом таких точек быть не должно, как и приглашения >>>.

Python считается языком с ясным синтаксисом и легко читаемым кодом. Это достигается сведением к минимуму таких вспомогательных элементов как скобок и точек с запятой. Для разделения выражений используется переход на новую строку, а для обозначения вложенных выражений – отступы от начала строки. В других языках данный стиль программирования также используется, но лишь для удобочитаемости кода человеком. В Питоне же он возведен в ранг синтаксического правила.

В примере выше логическим выражением является n < 100. Если оно возвращает истину, то выполнится строчка кода b = n + a. Если логическое выражение ложно, то выражение b = n + a не выполнится.

Данный пример вырван из контекста и сам по-себе не является рабочим. Полная версия программы могла бы выглядеть так:

b = 0

a = 50

n = 98

**if** n < 100:

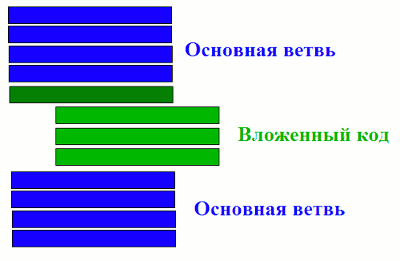
b = n + a

**print**(b)

Последняя строчка кода print(b) уже не относится к условному оператору, что обозначено отсутствием перед ней отступа. Она не является вложенной в условный оператор, значит, не принадлежит ему.

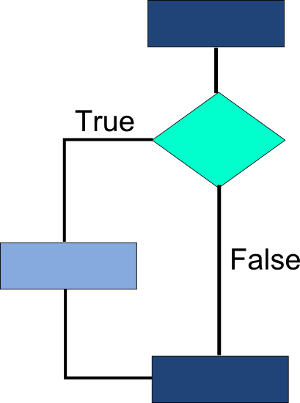
Поскольку переменная n равна 98, а это меньше 100, то b станет равной 148-ми. Это значение будет выведено на экран. Если переменная n изначально была бы связана, например, со значением 101, то на экран был бы выведен 0. При n, равной 101, логическое выражение в заголовке условного оператора вернуло бы ложь. Значит, тело не было бы выполнено, и переменная b не изменилась бы.

Структуру программы можно изобразить следующим образом:

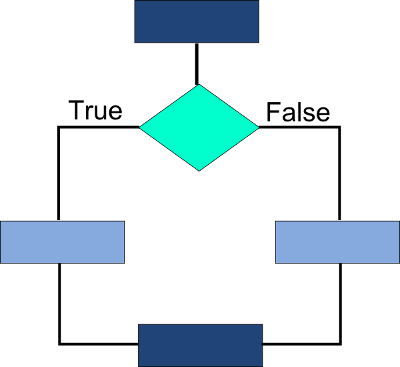


Основная ветка программы выполняется всегда, а вложенный код лишь тогда, когда в темно-зеленой строчке, обозначающей заголовок условного оператора, случается истина.

Для небольших программ иногда чертят так называемые блок-схемы, отражающие алгоритм выполнения. В языке блок-схем определенные конструкции обозначаются своими фигурами. Так блок действий обозначается прямоугольником, а логическое выражение – ромбом. Для кода выше блок-схема может выглядеть так:



Условный оператор может включать не одну ветку, а две, реализуя тем самым полноценное ветвление.



В случае возврата логическим выражением False поток выполнения программы не возвращается сразу в основную ветку. На случай False существует другой вложенный код, отличный от случая True. Другими словами, встретившись с расширенной версией условного оператора, поток выполнения программы не вернется в основную ветку, не выполнив хоть какой-нибудь вложенный код.

В языках программирования разделение на две ветви достигается с помощью добавления блока **else**, получается так называемое **if–else** (если-иначе). Синтаксис выглядит примерно так:

if логическое\_выражение {

выражение 1;

выражение 2;

…

}

else {

выражение 3;

…

}

Если условие при инструкции if оказывается ложным, то выполняется блок кода при инструкции else. **Ситуация, при которой бы выполнились обе ветви, невозможна.** Либо код, принадлежащий if, либо код, принадлежащий еlse. Никак иначе. **В заголовке else никогда не бывает логического выражения.**

Пример кода с веткой else на языке программирования Python:

tovar1 = 50

tovar2 = 32

**if** tovar1+ tovar2 > 99 :

**print**("99 рублей недостаточно")

**else**:

**print**("Чек оплачен")

Следует иметь в виду, что логическое выражение при if может выглядеть "нестандартно", т. е. не так просто, как a > b и тому подобное. Там может стоять просто одна переменная, число, слово True или False, а также сложное логическое выражение, когда два простых соединяются через логически И или ИЛИ.

a = ?

**if** a:

a = 1

Если вместо знака вопроса будет стоять 0, то с логической точки зрения это False, значит выражение в if не будет выполнено. Если a будет связано с любым другим числом, то оно будет расцениваться как True, и тело условного оператора выполнится. Другой пример:

a = 5 > 0

**if** a:

**print**(a)

Здесь a уже связана с булевым значением. В данном случае это True. Отметим, что в выражении a = 5 > 0 присваивание выполняется после оператора сравнения, так что подвыражение 5 > 0 выполнится первым, после чего его результат будет присвоен переменной a. На будущее, если вы сомневаетесь в последовательности выполнения операторов, используйте скобки, например так: a = (5 > 0).

Третий пример:

**if** a > 0 **and** a < b:

**print**(b - a)

Тут, чтобы вложенный код выполнился, a должно быть больше нуля и одновременно меньше b. Также в Питоне, в отличие от других языков программирования, позволительна такая сокращенная запись сложного логического выражения:

**if** 0 < a < b:

**print**(b - a)

# Множественное ветвление: if-elif-else

Ранее мы рассмотрели работу условного оператора if. С помощью его расширенной версии if-else можно реализовать две отдельные ветви выполнения. Однако алгоритм программы может предполагать выбор больше, чем из двух путей, например, из трех, четырех или даже пяти. В данном случае следует говорить о необходимости множественного ветвления.

Рассмотрим конкретный пример. Допустим, в зависимости от возраста пользователя, ему рекомендуется определенный видеоконтент. При этом выделяют группы от 3 до 6 лет, от 6 до 12, от 12 до 16, 16+. Итого 4 диапазона. Как бы мы стали реализовывать задачу, имея в наборе инструментов только конструкцию if-else?

Самый простой ответ – последовательно проверять вхождение введенного числа-возраста в определенный диапазон с помощью следующих друг за другом условных операторов:

old = int(input('Ваш возраст: '))

**print**('Рекомендовано:', end=' ')

**if** 3 <= old < 6:

**print**('"Заяц в лабиринте"')

**if** 6 <= old < 12:

**print**('"Марсианин"')

**if** 12 <= old < 16:

**print**('"Загадочный остров"')

**if** 16 <= old:

**print**('"Поток сознания"')

Примечание. Названия фильмов выводятся на экран в двойных кавычках. Поэтому в программе для определения строк используются одинарные.

Предложенный код прекрасно работает, но есть одно существенное "но". Он не эффективен, так как каждый if в нем – это отдельно взятый оператор, никак не связанный с другими if. Процессор тратит время и "нервы" на обработку каждого из них, даже если в этом уже нет необходимости. Например, введено число 10. В первом if логическое выражение возвращает ложь, и поток выполнения переходит ко второму if. Логическое выражение в его заголовке возвращает истину, и его тело выполняется. Всё, на этом программа должна была остановиться.

Однако следующий if никак не связан с предыдущим, поэтому далее будет проверяться вхождение значения переменной old в диапазон от 12 до 16, в чем необходимости нет. И далее будет обрабатываться логическое выражение в последнем if, хотя уже понятно, что и там будет False. Что же делать?

Ответом является вложение условных операторов друг в друга:

old = int(input('Ваш возраст: '))

**print**('Рекомендовано:', end=' ')

**if** 3 <= old < 6:

**print**('"Заяц в лабиринте"')

**else**:

**if** 6 <= old < 12:

**print**('"Марсианин"')

**else**:

**if** 12 <= old < 16:

**print**('"Загадочный остров"')

**else**:

**if** 16 <= old:

**print**('"Поток сознания"')

Рассмотрим поток выполнения этого варианта кода. Сначала проверяется условие в первом if (он же самый внешний). Если здесь было получено True, то тело этого if выполняется, а в ветку else мы даже не заходим, так как она срабатывает только тогда, когда в условии if возникает ложь.

Если внешний if вернул False, поток выполнения программы заходит в соответствующий ему внешний else. В его теле находится другой if со своим else. Если введенное число попадает в диапазон от 6 до 12, то выполнится тело вложенного if, после чего программа завершается. Если же число не попадает в диапазон от 6 до 12, то произойдет переход к ветке else. В ее теле находится свой условный оператор, имеющий уже третий уровень вложенности.

Таким образом до последней проверки (16 <= old) интерпретатор доходит только тогда, когда все предыдущие возвращают False. Если же по ходу выполнения программы возникает True, то все последующие проверки опускаются, что экономит ресурсы процессора. Кроме того, такая логика выполнения программы более правильная.

Теперь зададимся следующим вопросом. Можно ли как-то оптимизировать код множественного ветвления и не строить лестницу из вложенных друг в друга условных операторов? Во многих языках программирования, где отступы используются только для удобства чтения программистом, но не имеют никакого синтаксического значения, часто используется подобный стиль:

if логическое\_выражение {

… ;

}

else if логическое\_выражение {

… ;

}

else if логическое\_выражение {

… ;

}

else {

… ;

}

Может показаться, что имеется только один уровень вложенности, и появляется новое расширение для if, выглядящее как else if. Но это только кажется. На самом деле if, стоящее сразу после else, является вложенным в это else. Выше приведенная схема – то же самое, что

if логическое\_выражение {

… ;

}

else

if логическое\_выражение {

… ;

}

else

if логическое\_выражение {

… ;

}

else {

… ;

}

Именно так ее "понимает" интерпретатор или компилятор. Однако считается, что человеку проще воспринимать первый вариант.

В Питоне подобный номер с поднятием вложенного if к более внешнему else не прокатит, потому что в нем отступы и переходы на новую строку имеют синтаксическое значение. Поэтому в язык Python встроена возможность настоящего **множественного ветвления на одном уровне вложенности, которое реализуется с помощью веток elif**.

Слово "elif" образовано от двух первых букв слова "else", к которым присоединено слово "if". Это можно перевести как "иначе если".

В отличие от else, **в заголовке elif обязательно должно быть логическое выражение** также, как в заголовке if. Перепишем нашу программу, используя конструкцию множественного ветвления:

old = int(input('Ваш возраст: '))

**print**('Рекомендовано:', end=' ')

**if** 3 <= old < 6:

**print**('"Заяц в лабиринте"')

**elif** 6 <= old < 12:

**print**('"Марсианин"')

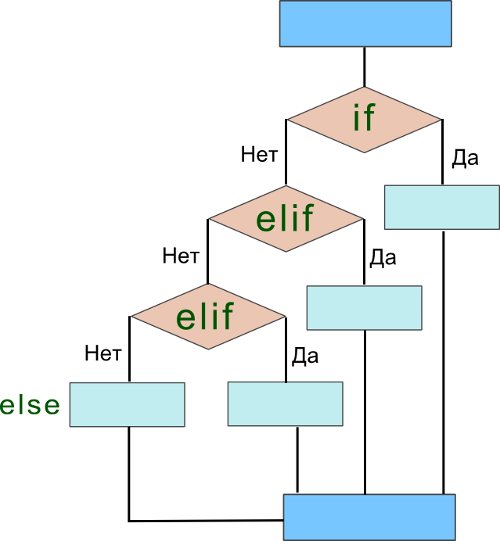
**elif** 12 <= old < 16:

**print**('"Загадочный остров"')

**elif** 16 <= old:

**print**('"Поток сознания"')

Обратите внимание, в конце, после всех elif, может использоваться одна ветка else для обработки случаев, не попавших в условия ветки if и всех elif. Блок-схему полной конструкции if-elif-…-elif-else можно изобразить так:



Как только тело if или какого-нибудь elif выполняется, программа сразу же возвращается в основную ветку (нижний ярко-голубой прямоугольник), а все нижеследующие elif, а также else пропускаются.