**Тема занятия №04: Управляющие выражения. Блоки, условия, циклы**

**Введение**

Наряду с арифметическими и прочими операциями, в программировании повсеместно используются операции сравнения, которые позволяют создавать логические выражения на основании сравнений, и логические операции, позволяющие комбинировать несколько логических выражений в одно.

Логические выражения — любые конструкции, результатом выполнения которых является True или False.

Человеческое доверие демонстрирует работу логических выражений. Каждый раз, когда мы слышим какое-либо высказывание, мы или верим ему (т. е. считаем истинным), или относимся к нему с недоверием (т. е. считаем ложным). Например, когда ваш друг позвал вас на прогулку и сказал, что на улице хорошая погода, может произойти две ситуации:

■вы посмотрите на улицу, убедитесь, что погода хорошая (утверждение друга истинно), и вы пойдете на прогулку;

■вы посмотрите на улицу, увидите, что погода плохая (утверждение друга ложно), и вы останетесь дома.

Конечно, в реальности мы можем сомневаться, однако в программировании оценка выражений всегда сводится к одному из этих двух вариантов. Для представления истинности и ложности используются ключевые слова True и False соответственно.

**4.1 Операции сравнения**

Операции сравнения используются для создания условий на основании сравнения элементов. Элементы, к которым применяется операция, называют операндами. Python поддерживает следующие операции сравнения:

== «равно»: истинно (True), если оба операнда равны, иначе — ложно False;

!= «не равно»: истинно (True), если оба операнда не равны, иначе — ложно False;

> «больше чем»: истинно (True), если первый операнд больше второго, иначе — ложно False;

< «меньше чем»: истинно (True), если первый операнд меньше второго, иначе — ложно False;

>= «больше или равно»: истинно (True), если первый операнд больше или равен второму, иначе — ложно False;

<= «меньше или равно»: истинно (True), если первый операнд меньше или равен второму, иначе — ложно False.

**print("1 == 1:", 1 == 1)**

**1 == 1: True**

**print("1 == 2:", 1 == 2)**

**1 == 2: False**

**print("1 != 1:", 1 != 1)**

**1 != 1: False**

**print("1 != 2:", 1 != 2)**

**1 != 2: True**

**print("1 > 1:", 1 > 1)**

**1 > 1: False**

**print("1 > 2:", 1 > 2)**

**1 > 2: False**

**print("2 > 1:", 2 > 1)**

**2 > 1: True**

**print("1 < 1:", 1 < 1)**

**1 < 1: False**

**print("1 < 2:", 1 < 2)**

**1 < 2: True**

**print("2 < 1:", 2 < 1)**

**2 < 1: False**

**print("1 >= 1:", 1 >= 1)**

**1 >= 1: True**

**print("1 >= 2:", 1 >= 2)**

**1 >= 2: False**

**print("2 >= 1:", 2 >= 1)**

**2 >= 1: True**

**print("1 <= 1:", 1 <= 1)**

**1 <= 1: True**

**print("1 <= 2:", 1 <= 2)**

**1 <= 2: True**

**print("2 <= 1:", 2 <= 1)**

**2 <= 1: False**

Выше представлен код, который демонстрирует результаты работы операций сравнения с целыми числами.

**4.2 Использование значений в качестве условий**

В Python существуют правила, согласно которым можно привести любое значение любого типа к логическому. Так, если вместо условия написать строку или число, программа заменит его на True или False.

Значения, которые будут заменены на False:

* структуры, не содержащие элементы:
* строки без символов;
* пустые списки, словари, массивы
* нули любых численных типов: 0; 0.0.
* константа None.

Значения, которые будут заменены на True:

структуры, содержащие элементы:

* строки любой длины, отличной от нуля;
* списки, словари, массивы и с элементами;
* ненулевые значения любых численных типов:

1, 2, 3…;

0.1, 1.2, 2.3… .

С помощью функции bool() можно проверить результат любого такого преобразования:

**print(bool(""))**

**False**

**print(bool(0.0))**

**False**

**print(bool(None))**

**False**

**print(bool("IT Step Academy"))**

**True**

**print(bool(1))**

**True**

Выше представлен код, который демонстрирует результаты работы операций сравнения с целыми числами.

**4.3 Логические операции**

Логические операции позволяют комбинировать несколько логических выражений в одно. В Python имеются следующие логические операторы:

* and

«Логическое умножение»: возвращает True, если оба выражения равны True. Например, работодатель, заинтересованный в хорошем сотруднике, возьмет человека на работу, если он компетентный И ответственный:

**competent = True responsible = True print(competent and responsible)**

**Результат работы нашего кода в консоли:**

**True**

Если же человек компетентен, но не ответственен:

**competent = True**

**responsible = False print(competent and responsible)**

**Результат работы нашего кода в консоли:**

**False**

* or

«Логическое сложение»: возвращает True, если хотя бы одно из выражений равно True. Например, работодатель, заинтересованный в найме сотрудника в кратчайшие сроки, возьмет человека на работу, если он компетентный ИЛИ ответственный:

**competent = True responsible = False**

**print(competent or responsible)**

**Результат работы нашего кода в консоли:**

**True**

Если же человек не компетентен и не ответственен:

**competent = False responsible = False**

**print(competent or responsible)**

**Результат работы нашего кода в консоли:**

**False**

* **not**

«Логическое отрицание»: возвращает значение, обратное операнду. Например, работодатель НЕ возьмет человека, который ранее был уволен:

**previously\_fired = True print(not previously\_fired)**

**Результат работы нашего кода в консоли:**

**False**

В противном случае:

**previously\_fired = False print(not previously\_fired)**

**Результат работы нашего кода в консоли:**

**True**

Обратите внимание, что если один из операндов оператора and возвращает False, то другой операнд не оценивается, так как результатом выполнения оператора в любом случае будет False. Аналогично, если один из операндов оператора or возвращает True, то второй операнд не оценивается, так как результатом выполнения оператора в любом случае будет True.

Рассмотрим работу логических операций на простом примере: возвращаясь домой, человек захотел воспользоваться метро. Он помнит, что метро работает с 6 утра и до 24 (или 0), для проезда ему необходимо предъявить билет или оплатить поездку, кроме того, его не пустят с большим багажом. На часы он еще не смотрел, билет забыл дома, денег на проезд у него хватает, и он без багажа:

**time = int(input("Enter the current time in hours: "))% 24**

**ticket = False**

**oney = True**

**luggage = False**

**print(money or ticket and not luggage and time > 6)**

На первый взгляд наше условие верно. Однако, оно истинно вне зависимости от значений time и luggage.

Классический пример из математики гласит, потому как сначала нужно умножить, а потом прибавить. Такой специфический порядок вычислений определен приоритетом операций. То же самое есть и у логических операций: сначала выполняется not, затем and, затем or.

Именно поэтому выполняющийся в последнюю очередь or вернет True. Как и с математическими операциями, для переопределения порядка выполнения можно использовать скобки**:**

**print((money or ticket) and not luggage and time > 6 )**

**Условный оператор**

На практике простейшим примером применения логических выражений является if: Если условие истинно — выполняется набор инструкций:

**car\_speed = 100**

**if car\_speed > 50:**

**print("Car is faster than 50 km/h")**

В этом случае результатом выполнения кода будет вывод на консоль строки «You have at least 3 flowers», т.к. выражение 2 < 3 истинно.

**passenger\_weight = 400 if passenger\_weight < 300:**

**print("The elevator can go")**

В этом примере лифт не сможет поехать, и строка «The elevator can go» не будет выведена на консоль, т.к. выражение 400 < 300 ложно.

Рассмотрим пример применения логических операций с if:

**car\_speed = 100**

**if car\_speed > 50 and car\_speed < 150:**

**print("Car speed is between 50 km/h and 150 km/h")**

В данном примере строка «Car speed is between 50 km/h and 150 km/h» будет выведена в случае, если переменная «car\_speed» находиться в промежутке от 50 до 150 (не включительно). Python позволяет записать такое условие подобно двойному неравенству в математике:

**if 50 < car\_speed < 150**:

Известно, что високосным годом считается тот год, который кратен 4, не кратен 100 или кратен 400. Запишем условное выражение, определяющее, является ли год високосным:

**year = 2000**

**if year % 4 == 0 and year % 100 != 0 or year % 400 == 0: print("Year", year, "is leap")**

Ранее был приведен простейший пример использования условной конструкции if:

**car\_speed = 100**

**if car\_speed > 50:**

**print("Car is faster than 50 km/h")**

Попробуем расширить его на основании полученных знаний. Введем две переменные, обозначающие скорость (в км/ч) машины и мотоцикла соответственно:

**car\_speed = 150**

**motorcycle\_speed = 100**

В таком случае наш пример можно изменить с использованием переменных:

**if car\_speed > motorcycle\_speed:**

**print("Car is faster than motorcycle")**

Результат работы нашего кода в консоли:

**Car is faster than motorcycle**

Если изменить скорость так, что машина окажется быстрее, наше условие окажется ложным и пользователь не получит никакой информации. В таком случае удобно будет использовать else — блок, который выполняется только если последнее условие оказалось ложным. Добавим его в наш пример:

**car\_speed = 100**

**motorcycle\_speed = 150**

**if car\_speed > motorcycle\_speed:**

**print("Car is faster than motorcycle")**

**else:**

**print("Motorcycle is faster than car")**

Результат работы нашего кода в консоли:

**Motorcycle is faster than** **car**

Скорости могут уравняться и, хотя блок else выполнится, представленная пользователю информация будет неверна. Исправить это можно, добавив еще одно условие:

**car\_speed = 100**

**motorcycle\_speed = 100**

**if car\_speed > motorcycle\_speed:**

**print("Car is faster than motorcycle")**

**if motorcycle\_speed > car\_speed:**

**print("Motorcycle is faster than car")**

**else:**

**print("Car and motorcycle are equally fast")**

Результат работы нашего кода в консоли:

**Car and motorcycle are equally fast**

Здесь условные конструкции выполняются поочередно: сначала проверяется условие motorcycle\_speed < car\_speed, затем motorcycle\_speed > car\_speed. Обратите внимание, что второе условие будет проверено вне зависимости от истинности первого. Полезность подобного поведения ситуативная: если эти конструкции самостоятельны и предполагают, как независимое, так и совместное выполнение — это полезно, однако если они связаны и предполагается выполнение только одной конструкции — это может привести к ошибкам.

Данный пример хорошо справляется со сравнением постоянных скоростей. Если же за время сравнения скорость мотоцикла изменится, мы получим два противоречивых результата:

**car\_speed = 130**

**motorcycle\_speed = 100**

**if car\_speed > motorcycle\_speed:**

**print("Car is faster than motorcycle")**

**motorcycle\_speed += 50**

**if motorcycle\_speed > car\_speed:**

**print("Motorcycle is faster than car")**

**motorcycle\_speed += 50**

**else:**

**print("Car and motorcycle are equally fast")**

**motorcycle\_speed += 50**

Результат работы нашего кода в консоли:

**Car is faster than motorcycle**

**Motorcycle is faster than car**

Так, при первом сравнении условие motorcycle\_speed < car\_speed будет истинно, скорость мотоцикла изменится, и при втором сравнении motorcycle\_speed > car\_speed также окажется истинным.

Избежать подобной ситуации можно, объединив два условия в одну конструкцию. Для этого заменим второй if на elif — комбинацию блоков else и if, набор инструкций которого выполняется если условия предыдущих блоков ложны, а собственное условие блока — истинно:

**if car\_speed > motorcycle\_speed:**

**print("Car is faster than motorcycle")**

**motorcycle\_speed += 50**

**elif motorcycle\_speed > car\_speed:**

**print("Motorcycle is faster than car")**

**motorcycle\_speed += 50**

**else:**

**print("Car and motorcycle are equally fast")**

**motorcycle\_speed += 50**

Результат работы нашего кода в консоли:

**Car is faster than motorcycle**

**Блоки в PYTHON**

В Python, в отличие от многих других языков, инструкции, принадлежащие к одному блоку выполнения, достаточно выделить одинаковым отступом. Концом блока выполнения считается последняя инструкция, которая будет выделена соответствующим отступом.

В играх подобным образом определяются комбинации или «комбо»: успешные действия, идущие друг за другом, образуют «комбо», которое прерывается, как только действие будет выполнено неудачно.

Рассмотрим пример блока выполнения:

**if 1 > 4: # line 1**

**print("This is the start of an execution block") # line 2**

**print("This is part of the execution block") # line 3**

**print("This is still part of the execution block") # line 4**

**print("This is the end of an execution block") # line 5**

**print("It is not part of the execution block") # line 6**

В этом случае результатом выполнения кода будет вывод на консоль строки «It is not part of the execution block», т. к. выражение 1 > 4 ложное и инструкции, принадлежащие к блоку выполнения конструкции, не будут выполнены.

В частности, к нему принадлежат строки от 2 до 5. Отступы должны состоять из пробелов и, согласно руководству по оформлению кода, их количество желательно делать кратным 4 (4 пробела, 8, 16…). Хотя если блок будет выделен отступом в 2 пробела, программа также будет работать.

Кроме того, в последних версиях языка для отступа нельзя использовать табуляцию, однако современные среды разработки (в сокращенном варианте «IDE») автоматически размещают 4 пробела при нажатии на клавиатуре кнопки «Tab».

**Применение**

Определение блоков выполнения необходимо не только при использовании условных конструкций. Они также используются в:

* циклах;

**while condition:**

**print("Cycles")**

* обработке исключений;

**try:**

**print("Some**

**code")**

**except:**

**print("Error processing")**

**finally:**

**print("Handling Exceptions")**

* функциях.

**def function():**

**while condition:**

**print("Cycles")**

**print("Functions")**

**Условия в PYTHON. Практика**

**Модуль math**

Модуль math – один из наиважнейших в Python. Этот модуль предоставляет обширный функционал для работы с числами.

Ниже представлены методы модуля math:

**math.ceil**(X) – округление до ближайшего большего числа.

**math.copysign**(X, Y) - возвращает число, имеющее модуль такой же, как и у числа X, а знак - как у числа Y.

**math.fabs**(X) - модуль X.

**math.factorial**(X) - факториал числа X.

**math.floor**(X) - округление вниз.

**math.fmod**(X, Y) - остаток от деления X на Y.

**math.frexp**(X) - возвращает мантиссу и экспоненту числа.

**math.ldexp**(X, I) - X \* 2i. Функция, обратная функции math.frexp().

**math.fsum**(последовательность) - сумма всех членов последовательности. Эквивалент встроенной функции sum(), но math.fsum() более точна для чисел с плавающей точкой.

**math.isfinite**(X) - является ли X числом.

**math.isinf**(X) - является ли X бесконечностью.

**math.isnan**(X) - является ли X NaN (Not a Number - не число).

**math.modf**(X) - возвращает дробную и целую часть числа X. Оба числа имеют тот же знак, что и X.

**math.trunc**(X) - усекает значение X до целого.

**math.exp**(X) - eX.

**math.expm1**(X) - eX - 1. При X → 0 точнее, чем math.exp(X)-1.

**math.log**(X, [base]) - логарифм X по основанию base. Если base не указан, вычисляется натуральный логарифм.

**math.log1p**(X) - натуральный логарифм (1 + X). При X → 0 точнее, чем math.log(1+X).

**math.log10**(X) - логарифм X по основанию 10.

**math.log2**(X) - логарифм X по основанию 2.

**math.pow**(X, Y) - XY.

**math.sqrt**(X) - квадратный корень из X.

**math.acos**(X) - арккосинус X. В радианах.

**math.asin**(X) - арксинус X. В радианах.

**math.atan**(X) - арктангенс X. В радианах.

**math.atan2**(Y, X) - арктангенс Y/X. В радианах. С учетом четверти, в которой находится точка (X, Y).

**math.cos**(X) - косинус X (X указывается в радианах).

**math.sin**(X) - синус X (X указывается в радианах).

**math.tan**(X) - тангенс X (X указывается в радианах).

**math.hypot**(X, Y) - вычисляет гипотенузу треугольника с катетами X и Y (math.sqrt(x \* x + y \* y)).

**math.degrees**(X) - конвертирует радианы в градусы.

**math.radians**(X) - конвертирует градусы в радианы.

**math.cosh**(X) - вычисляет гиперболический косинус.

**math.sinh**(X) - вычисляет гиперболический синус.

**math.tanh**(X) - вычисляет гиперболический тангенс.

**math.acosh**(X) - вычисляет обратный гиперболический косинус.

**math.asinh**(X) - вычисляет обратный гиперболический синус.

**math.atanh**(X) - вычисляет обратный гиперболический тангенс.

**math.erf**(X) - функция ошибок.

**math.erfc**(X) - дополнительная функция ошибок (1 - math.erf(X)).

**math.gamma**(X) - гамма-функция X.

**math.lgamma**(X) - натуральный логарифм гамма-функции X.

**math.pi** - pi = 3,1415926...

**math.e** - e = 2,718281...