Поток управления: введение

Каждый день мы сталкиваемся с проблемой выбора в зависимости от условий. Например, каким образом поехать на работу? Если общественным транспортом, то проблема с тем, что мы едем с большим количеством людей, зато будем на месте по расписанию. Если ехать на собственном автомобиле, то сама поездка будет гораздо комфортнее, но можем столкнуться с проблемой поиска парковки. Такие проблемы решаются с использованием правил «Если, то». Например, если на улице холодно, то нужно надеть куртку и т.д. Иными словами, некоторый порядок наших действий зависит от того, выполняется ли какое-нибудь условие.

В python последовательность действий называется набор команд, которые следуют в определенном порядке. Последовательность исполнения команд называется поток. До настоящего времени мы сталкивались только с прямым исполнением команд, то есть построчно сверху вниз. Но иногда нам необходимо некоторое ветвление потока исполнения, то есть выполнение некоторых команд только в том случае, если выполнено какое-то условие. Для этих целей используется составной оператор if/else.

В рамках данной темы вы научитесь управлять потоком исполнения команд с использованием логических выражений и оператора if/else.

Операторы сравнения (логические операторы)

Разберемся, что такое логические выражения. Под логическим выражением мы понимаем высказывание, которое задает какое-то условие. Например, выражение «На улице идет дождь». Данное выражение может быть либо правдой, либо ложью. В зависимости от того, есть ли дождь, принимается решение о необходимости зонтика. То есть логическое выражение может быть либо истинным, либо ложным.

Теперь, стало понятно, что такое логические выражения, необходимо научиться создавать их на Python. Мы можем создать логическое выражение, используя операторы сравнения.

Из названия видно, что данные операторы сравнивают 2 значения и возвращают либо True (если истина), либо False (если ложно).

Существует несколько видов операторов отношения. Для начала рассмотрим простой случай:

• Равно: == (обратите внимание, что равно состоит из двух знаков ==)

• Не равно:! =

Эти операторы сравнивают два элемента и возвращают True или False, если они равны или нет.

Мы можем создавать логические выражения, сравнивая два значения с помощью этих операторов:

```
>>>print (1 == 1)
True
>>>print(2 != 4)
True
>>> print(3 == 5)
```

```
False
>>> print('7' == 7)
False
```

Выше приведены примеры логических выражений. Как видно из примеров, результат исполнения команды, содержащей логическое выражение, является True или False.

Обратите внимание! True и False – это отдельный логический тип данных. До сих пор мы рассматривали только тип строка (string), тип целые числа (int) и тип числа с плавающей запятой (float). Логический тип называется **bool** и принимает только 2 вида значений: True или False. Обратите внимания, что данные значения пишутся с **заглавной** буквы. Напомним, что python чувствителен к регистру и True ≠true. Можно заметить, что при вводе их в редакторе кода (с заглавными буквами T и F), они отображаются цветом, отличным от переменных или строк.

Задание:

Определите, являются ли следующие логические выражения истинными или ложными.

Утверждение первое:

```
(6 * 6) - 1 == 8 + 1
```

Утверждение второе:

```
13 - 7 != (3 * 2) + 1
```

Утверждение третье:

```
3 * (2 - 1) == 4 - 1
```

Логические операторы больше и меньше.

Выше мы рассмотрели 2 типа операторов сравнения: равно и не равно. Кроме них существуют и другие операторы сравнения:

• Больше чем:>

• Меньше чем: <

• Больше или равно:> =

• Меньше или равно: <=

Задание:

Определите, являются ли следующие логические выражения истинными или ложными.

```
(6 * 6) - 1 >= 8 + 1
```

Утверждение второе:

```
13 - 7 <= (3 * 2) + 1
```

Утверждение третье:

```
3 * (2 - 1) > 4 - 1
```

Логические переменные

Так как True и False являются отдельным особым типом данных bool, то соответственно, как и в случае с числами и строками, можно создавать переменные, содержащие значения истины или лжи.

True и False - единственные типы bool, и любая переменная, которой присвоено одно из этих значений, называется логической переменной. Логические переменные можно создавать несколькими способами. Самый простой способ - просто присвоить переменной True или False:

```
>>true_condition = True
>>false_condition = False
```

Можно установить переменную, равную логическому выражению.

```
>>bool_condition_1 = 5 != 7
>>bool_condition_2 = 1 + 1 != 2
>>bool_condition_3  = 3 * 3 == 9
```

Эти переменные теперь содержат логические значения, поэтому при ссылке на них они будут возвращать только значения True или False того выражения, которое им было присвоено.

```
>>>print (bool_three)
True
>>> print(bool_four)
False
>>>print(bool_five)
True
```

Задание:

1. Создайте переменную с именем bool_variable и установите для нее значение true. Попробуйте вывести ее в консоль. Какая ошибка у вас вывелась? Почему?

2. Замените значение в bool_variable на ‘true’ (в кавычках) Проверьте тип bool_variable, используя функцию проверки типов type (bool_variable).

Выведите значение в консоль. Какой тип теперь у переменной bool_variable?

3. Почему это не логическая переменная!

Создайте переменную с именем bool_variable_2 и сделайте так, чтобы она имелв логический тип истины.

4. Проверьте тип bool_variable_2 и убедитесь, что вы успешно создали логическую переменную.

Выражения if.

Понимание логических переменных и выражений важно, потому что они являются строительными блоками при составлении проверочных утверждений с использованием условных операторов.

Вернемся к примеру с дождем.

```
Если пойдет дождь, то вы возьмете зонт
```

Если перевести на английский язык, то получим следующее утверждение:

```
If it rains, then you take an umbrella
```

Тогда «пойдет дождь» - это логическое выражение, и этот условный оператор проверяет, истинно ли оно.

Если "идет дождь" == Истина, то остальная часть условного оператора будет выполнена, и вы принесете зонтик.

На самом деле, это форма условного оператора:

```
If [it rains] then [take an umbrella]
```

Условный оператор if проверяет истинность выражения, следующим за ним. Язык программирования Python хорош тем, что очень близок к конструкциям естественного английского языка.

В Python проверка условия будет выглядит очень похоже на пример с зонтиком, сформулированном на английском языке:

```
>>if is_raining:
>>  bring_umbrella()
```

Обратите внимание, что вместо слова «то» после условия используется двоеточие «:». Это интерпретируется компьютером так, что дальше будет тот набор действий (команд), которые должны быть выполнены в случае истинности условия.

Обратите внимание! Действия, которые выполняются в случае истинности условия должны быть выделены отступами, причем, их должно быть одинаковое количество. То есть сам if мы пишем от начала строки, а все, что выполняется в случае истинности, имеют одинаковый отступ от края строки. Например:

```
>>age = 13
>>if age >=18:
>>      print('Добро пожаловать на сайт алкогольной продукции')
>>print('проверка выполнена')
```

В данном примере мы видим, что задается переменная age, далее проверяется условие, age больше либо равен 18. Если это истина, то тогда вывести в консоль строку «Добро пожаловать на сайт алкогольной продукции». Однако, переменная age равна 13,

условие в if ложно, соответственно, строга не будет выведена в консоль, однако будет выведена строка «проверка выполнена», так как она является частью основного кода и не входит в оператор if. Это связано с тем, что последний print написан от начала строки и не имеет отступов, то есть будет выполняться в любом случае вне зависимости от проверки условия в операторе if. Результатом исполнения данного кода будет:

```
проверка выполнена
```

Давайте посмотрим на другой условный оператор:

```
>>if 10 == 5 * 2:
>>  print("выражение это значение истинное имеет")
```

Будет ли этот код выводить выражение это значение истинное имеет на терминал? Да, потому что условие оператора if `10 == 5 * 2` истинно.

Задание:

Я работаю в компании, обслуживающей проблемы информационной безопасности на других предприятиях. К нам обратился директор маленькой рекламной компании. Основная проблема в том, что у них есть охранник Дмитрий, который устанавливает компьютерные игры на АРМ (автоматизированные рабочие места) сотрудников, когда тех нет на работе по долгу (находятся в отпуске) и играет всеми ночами напролет. Поэтому вас просят разработать приложение, которое проверяло введенные учетные и разграничивало права сотрудников. Для охранника Дмитрия просят сделать специализированное уведомление: «Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!».

1. Введем переменную «user_name»
2. Далее введем переменную, которая выводит текст для Дмитрия. Назовем ее Dmitriy_check
3. Введем переменную, которая хранит сообщение для других сотрудников, вошедших в систему: «Добро пожаловать»
4. Напишем оператор if, которые проверяет значение переменной user_name
5. Вывести в консоль результат выполнения программы для user_name= «Дмитрий» и для user_name= «Ангелина».

Наша компания продолжает разрабатывать приложение по безопасности для рекламной компании. Теперь, если пользователь 3 раза ввел пароль неправильно, необходимо заблокировать систему. Для этого выполним следующие действия:

1. Введем переменную, которая фиксирует количество попыток ввода и назовем ее enter_number
2. С использованием оператора if напишем программу, которая если enter_number меньше 3, то пишите «Попробуйте еще раз. У вас осталось (3- enter_number) попыток». Если количество попыток больше либо равно 3, то выводим «Вы превысили максимальное число попыток. Ваша учетная запись заблокирована. Для разблокировки обратитесь в службу поддержки».
3. Проверить работу программы с использованием вывода в консоль.

Логические операторы: and

Мы рассмотрели операторы сравнения. Они являются разновидностью логических операторов. Существуют и другие типы логических операторов.

Существуют ситуации, при которых необходимо проверять сразу несколько условий. Например, мы берем зонтик только в том случае, если собираемся выходить на улицу в дождь. То есть одновременно должно быть 2 истинных условия:

1. На улице идет дождь
2. Мне нужно выйти на улицу

Для построения более сложных условия используются логические операторы:

- `and`
- `or`
- `not`

Начнем с `and`.

`and` проверяет выполнение двух логических выражений одновременно (логическое и) и примет значение True, если оба его логических выражения имеют значение True, и значение False в противном случае.

Рассмотрим пример

```
Курица – это птица и панда – это животное.
```

Это логическое выражение состоит из двух меньших выражений: курица - птица, а панда - животное, оба из которых имеют значение `True` и связаны логическим оператором `and (и)`, поэтому все выражение истинно.

Давайте посмотрим на пример некоторых оператора `and` в Python:

```
>>> (1 + 1 == 2) and (2 + 2 == 4)
True
>>> (1 + 1 == 2) and (2 < 1)
False
>>> (1 > 9) and (5 != 6)
False
>>> (0 == 10) and (1 + 1 == 1)
False
```

Обратите внимание, что во втором и третьем примерах, даже если часть выражения имеет значение True, все выражение в целом имеет значение False, потому что другой оператор имеет значение False. Четвертый оператор также имеет значение False, поскольку оба компонента имеют значение False.

Задание:

1. Проверить истинность следующих выражений:

```
(2 + 2 + 2 >= 6) and (-1 * -1 < 0)
(4 * 2 <= 8) and (7 - 1 == 6)
```

Результат проверки поместить в переменные statement_one и statement_two.

2. Вернемся к задаче с проверкой безопасности в рекламном агентстве. Усложним задачу. Теперь нужно проверять не только имя пользователя, но и номер АРМ. Для этого введем номера рабочих мест 4х сотрудников этой компании:
   Дмитрий номер АРМ 1
   Ангелина номер АРМ 2
   Василий номер АРМ 3
   Екатерина номер АРМ 4.
3. Введем переменную ARM, обозначающую номер АРМ.
4. Создадим условный оператор if, при котором проверяем соответствие номера АРМ и имени пользователя:

Если номер АРМ и имя пользователя соответствуют, то вывести в консоль «Добро пожаловать!»

Если номер АРМ не совпадает, а имя пользователя не Дмитрий, то «Логин или пароль не верный, попробуйте еще раз»

Если Если номер АРМ не совпадает, а имя пользователя Дмитрий, то Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!».

Логические операторы: or

Логический оператор or (логическое или) подразумевает, что выполняется хотя бы одно условие, то есть имеет значение True, если любой из компонентов имеет значение True.

Рассмотрим выражение:

```
Курица – это птица или панда – это рыба.
```

Это утверждение состоит из двух выражений: курица - птица, которое является истинным, а панда - рыба, которое является ложным. Поскольку два выражения связаны оператором or, весь оператор является истинным. Только один компонент должен быть истинным, чтобы оператор или был истинным.

Если оператор or имеет два компонента True, он также является True.

Давайте взглянем на пару примеров на Python:

```
>>> True or (3 + 4 == 7)
True
>>> (1 - 1 == 0) or False
True
>>> (2 < 0) or True
True
>>> (3 == 8) or (3 > 4)
False
```

Обратите внимание, что каждый оператор or, имеющий хотя бы один компонент True, имеет значение True, но последний оператор имеет два компонента False, поэтому он равен False.

Задание:

1. Проверить истинность следующих выражений:

```
(2 - 1 > 3) or (-5 * 2 == -10)
(9 + 5 <= 15) or (7 != 4 + 3)
```

Выражение else

Как вы можете видеть из своей работы с задачей по информационной безопасности, как только вы начинаете включать в функцию множество операторов if, код становится немного загроможденным и неуклюжим. К счастью, есть и другие инструменты, которые мы можем использовать для управления потоком исполнения.

Операторы else позволяют описать, что необходимо сделать в случае, когда условия **не** выполняются.

Операторы else всегда появляются вместе с операторами if. Рассмотрим наш пример пробуждения, чтобы увидеть, как это работает:

```
>>if weekday:
>>  wake_up("6:30")
>>else:
>>  sleep_in()
```

Таким образом, мы можем создавать операторы if, которые позволяют выбирать, какие команды будут исполняться при условии истинности условий, а какие при условии ложности. Это избавляет нас от необходимости писать операторы if для каждого возможного условия, вместо этого мы можем написать общий оператор else для всех случаев, когда условие не выполняется.

Вернемся к примеру с сайтом алкогольной продукции. Давайте доработаем код так, чтобы в случае, если возраст меньше 18, выводил сообщение «Приносим наши извинения, мы не продаем продукцию лицам моложе 18 лет». Тогда наш код будет выглядеть следующим образом:

```
>>age = 13
>>if age >=18:
>>      print('Добро пожаловать на сайт алкогольной продукции')
>>else:
>>      print('Приносим наши извинения, мы не продаем продукцию лицам моложе 18 лет')
```

Обратите внимания на отступы! else располагается на одном уровне с if, а условия, выполняемые в случае ложности условия так же имеют отступ от начала строки (то есть выделяется тот блок кода, который будет выполнен в случае, если возраст меньше 18).

Задания:

Провести рефакторинг (переделать код) в задаче про Дмитрия и рекламное агентство с использованием оператора else.

Else if оператор

Помимо оператора if и оператора else в python существует оператор elif.

Давайте разберемся, как это работает. Предположим, мы хотим сходить в кафе и купить какой-нибудь сэндвич. Мы каждый день туда ходим и четко знаем, что там продается и, конечно, у нас уже есть вкусовые предпочтения. Больше всего мы любим сэндвич с курицей, на втором месте сэндвич с говядиной, на последнем сэндвич с тунцом. В случае отсутствия всех перечисленных возьмем что-то другое. Итак, попробуем написать код с использованием оператора if / else. Введем переменную, содержащую значение того типа сэндвича, который остался в наличии в магазине.

```
>> sandwich_in_stock = ‘sandwich_
>>if sandwich_in_stock == ‘с курицей’:
>>      print(‘куплю сэндвич с курицей’)
>>else:
>>      if sandwich_in_stock == ‘с говядиной’:
>>              print(‘куплю сэндвич с говядиной’)
>>      else:
>>              if sandwich_in_stock == ‘с тунцом’:
>>                      print(‘куплю сэндвич с тунцом’)
>>              else:
>>                      print(‘возьму что-нибудь другое)
```

Из этого примера видно, что получилась сложная и сложно читаемая конструкция. Если условие ложно, то внутри оператора else приходится проверять еще дополнительные условия. Это называется вложенные операторы. Обратите внимание, что чем дальше вложен оператор, тем бОльший отступ от начала строки.

Для того, чтобы не прописывать вложенный оператор if в python введен специальный оператор elif.

То есть оператор elif - это именно то, на что похоже «иначе, если». Оператор elif проверяет другое условие после предыдущего условия if, которое не выполняется.

Можно использовать операторы elif для управления порядком, в котором наша программа должна проверять каждый из наших условных операторов. Сначала проверяется оператор if, затем каждый оператор elif проверяется сверху вниз, а затем, наконец, выполняется код else, если ни одно из предыдущих условий не было выполнено.

Давайте переделаем код с сэндвичами и переделаем его с использованием оператора elif.

```
>>if sandwich_in_stock == 'с курицей':
>>      print('куплю сэндвич с курицей')
>>elif sandwich_in_stock == 'с говядиной':
>>      print('куплю сэндвич с говядиной')
>>elif sandwich_in_stock == 'с тунцом':
>>      print('куплю сэндвич с тунцом')
>>else:
>>      print('возьму что-нибудь другое')
```

Рассмотрим другой пример. Предположим, разрабатывается сайт для сбора пожертвований и в зависимости от того, какую сумму вы жертвуете, вам выводится соответствующее сообщение:

```
>>if donation >= 1000:
>>    print("Спасибо за ваше пожертвование! Вы получили платиновый статус пожертвования!
>>elif donation >= 500:
>>    print("Спасибо за ваше пожертвование! Вы получили золотой статус пожертвования!")
>>elif donation >= 100:
>>    print("Спасибо за ваше пожертвование! Вы получили серебряный статус пожертвования!")
>>else:
>>    print("Спасибо за ваше пожертвование! Вы получили бронзовый статус пожертвования!")")
```

Подумайте об этой функции на секунду. Что бы произошло, если бы все операторы elif были просто операторами if? Если бы было пожертвовано 1000, то все первые три сообщения были бы выведены в консоль, потому что каждое из условий было выполнено.

Но поскольку мы использовали операторы elif, он последовательно проверяет каждое условие и печатает только одно сообщение. Если я пожертвую 600,00, код сначала проверяет, превышает ли это 1000,00, а это не так, затем он проверяет, превышает ли это 500,00, что так и есть, поэтому выводится соответствующее сообщение, а затем, поскольку все другие утверждения проверяются только при условии ложности предыдущего, то они не проверяются и сообщения больше не выводятся.

Задание:

В университете действует система грейдов, которая присваивается студенту в зависимости от среднего балла. Вас просят написать приложение, которое выводило бы соответствующий грейд для каждого студента при следующих условиях:

- 4.0 или выше должен вернуть "A"
- 3.0 или выше должен вернуть "B"
- 2.0 или выше должен вернуть "C"
- 1.0 или выше должен вернуть "D"
- 0,0 или выше должен вернуть "F"

Для решения ввести переменную grade, а также использовать оператор elif.

Операторы Try и Except

Операторы if, elif и else - не единственный способ встроить поток управления в вашу программу. Вы можете использовать операторы try и except для проверки возможных ошибок, с которыми может столкнуться пользователь.

Общий синтаксис операторов try и except:

```
>>try:
>>    # some statement
>>except ErrorName:
>>     # some statement
```

Во-первых, будет выполнен оператор try. Если в какой-то момент во время этого выполнения возникает исключение, такое как NameError или ValueError, и это исключение соответствует ключевому слову в операторе except, тогда оператор try будет завершен, и оператор except будет выполнен.

На самом деле, ошибки можно проверять и с использованием if/else.

Существует 2 подхода к управлению потока:

EAFP (easier to ask forgiveness than it is to get permission, перевод: проще попросить прощения, чем получить разрешение). При этом подходе мы применяем конструкцию try/except. В этом случае при выполнении программы сначала производится попытка что-то сделать (например, осуществить деление), а в случае возникновения ошибки выполняется другой код.

Давайте посмотрим на это на примере. Необходимо написать код, который осуществляет деление a на b. Но есть вероятность, что b равно нулю, что вызовет ошибку, поэтому необходимо включить try и except, чтобы отловить эту ошибку.

```
>>try:
>>    result = a / b
>>    print (result)
>>except ZeroDivisionError:
>>    print ("деление на ноль не допустимо ")
```

LBYL (Look before you leap, перевод: смотри, прежде чем прыгать). При этом подходе сначала проверяется, что значения допустимы, а потом выполняется код.

Пример с делением будет выглядеть так:

```
>>if b!=0:
>>      result = a/b
>>      print(result)
>>else:
>>      print('деление на ноль не допустимо')
```

Выводы

В этом разделе мы рассмотрели много новых инструментов языка Python, которые позволяют нам добавить логики в наш код. Подведем итог. В рамках данного раздела было рассмотрено:

• Логические выражения - это утверждения, которые могут иметь значение True или False.

• Логическая переменная - это переменная, для которой установлено значение True или False.

• Вы можете создавать логические выражения, используя операторы сравнения:

o Равно: ==

o Не равно:! =

o Больше чем:>

o Больше или равно:> =

o Меньше чем: <

o Меньше или равно: <=

• операторы if могут использоваться для создания потока управления в вашем коде.

• Операторы else могут использоваться для выполнения кода, когда не выполняются условия оператора if.

• операторы elif могут использоваться для встраивания дополнительных проверок в операторы if.

• Операторы try и except могут использоваться для встраивания контроля ошибок в ваш код.