Лекция 5: Поток управления

Содержание

- Добро пожаловать на этот урок по Control Flow! Поток управления - это последовательность выполнения кода. Здесь мы узнаем о нескольких инструментах в Python, которые мы можем использовать для воздействия на поток управления нашего кода:
- Условные операторы
- Логические выражения
- Циклы For и While
- Операторы Break и Continue
- Функция Zip и Enumerate
- Генератора списка

Поток управления

- Условные операторы
- Оператор **if** - это условный оператор, который запускает или пропускает код в зависимости от того, является ли условие истинным или ложным. Вот простой пример:

```
if phone_balance < 5:
    phone_balance += 10
    bank_balance -= 10
```

- Давайте разберем его.
  - Оператор **if** начинается с ключевого слова **if**, за которым следует проверяемое условие, в этом случае **баланс _ телефона <5**, а затем двоеточие. Условие указывается в логическом выражении, которое оценивается как истина или ложь.
  - После этой строки следует блок кода с отступом, который выполняется, если это условие истинное. Здесь строки, которые увеличивают **баланс _ телефона** и уменьшают **остаток _ счета**, выполняются, только если верно, что **баланс _ телефона** меньше 5. Если нет, код в этом блоке if просто пропускается.

Использование операторов сравнения в условных выражениях

- Вы узнали об операторах сравнения Python (например, == и **!=**), чем они отличаются от операторов присваивания (например, =).
- В условных выражениях вы хотите использовать операторы сравнения.
- Например, вы хотите использовать **if x == 5,** а не **if x = 5**.
- Если ваш условный оператор вызывает синтаксическую ошибку или делает что-то неожиданное, проверьте, не написали ли вы == или = !

If

- В дополнение к выражению **if**, есть два других дополнительных выражения, часто используемых с оператором **If**. Например:

```
if season == 'spring':
    print('plant the garden!')
elif season == 'summer':
    print('water the garden!')
elif season == 'fall':
    print('harvest the garden!')
elif season == 'winter':
    print('stay indoors!')
else:
    print('unrecognized season')
```

- **if**: Оператор **if** всегда должен начинаться с выражения **if**, которое содержит первое проверенное условие. Если это оценивается как истина, Python запускает код с отступом в этом блоке **if**, а затем переходит к остальной части кода после оператора **if**.

Elif, Else

- **elif**: **elif -** это сокращение "else if." Выражение **elif** используется для проверки дополнительного условия, если условия в предыдущих выражениях в операторе **if** ложные. Как вы можете видеть в примере, у вас может быть несколько блоков **elif** для различных ситуаций.
- **else**: Последнее это выражение **else**, которое следует в конце оператора **if,** если используется. Это выражение не требует условия. Код в блоке **else** выполняется, если все условия выше, которые в операторе **if**, ложные.

Сложные логические выражения

- Операторы **if** иногда используют сложные логические выражения для своих условий. Они могут содержать несколько операторов сравнения, логические операторы и даже вычисления. Примеры:

```
if 18.5 <= weight / height**2 < 25:
    print("BMI is considered 'normal'")

if is_raining and is_sunny:
    print("Is there a rainbow?")

if (not unsubscribed) and (location == "USA" or location == "CAN"):
    print("send email")
```

Сложные логические выражения

- Для действительно сложных условий вам может потребоваться объединить некоторые операторы **and**, **or** и **not**. Используйте скобки, если вам нужно ясно показать комбинации.
- Однако простое или сложное - это выражение, условие в операторе **if** должно быть логическим выражением, которое оценивается как истина или ложь, именно это значение определяет, будет ли выполнен блок с отступом в операторе **if**.

Хорошие и плохие примеры

- Вот несколько вещей, о которых следует помнить при написании логических выражений для ваших операторов **if**.

1. Не используйте **истину** или **ложь** в качестве условий.

```
# Bad example
if True:
    print("This indented code will always get run.")
```

Несмотря на то, что «истина» является допустимым логическим выражением, оно не используется как условие, поскольку всегда оценивается как истина, поэтому код с отступом всегда запускается. Точно так же, если истина не является условием, которое вы должны использовать, выражение, следующее за оператором **if**, никогда не будет выполнено.

Хорошие и плохие примеры

```
# Another bad example
if is_cold or not is_cold:
    print("This indented code will always get run.")
```

- Точно так же бесполезно использовать любое условие, которое, как вы знаете, всегда оценивается как истина, как в примере выше.
- Логическая переменная может быть только истиной или ложью, поэтому **is_cold** или **not is_cold** всегда истина, и код с отступом всегда будет выполняться.

2. Будьте осторожны при написании выражений, которые используют логические операторы

- Логические операторы **and**, **or** и **not** имеют определенные значения, которые не совсем совпадают с их значением на разговорном английском языке. Убедитесь, что ваши логические выражения оцениваются так, как вы ожидаете.

```
# Bad example
if weather == "snow" or "rain":
    print("Wear boots!")
```

- Этот код допустимый в Python, но он не является логическим выражением, хотя считывается как один.
- Причина в том, что выражение справа от оператора **or**, **«rain»** не является логическим выражением - это строка!
- Позже мы обсудим, что происходит, когда вы используете нелогические объекты вместо логических.

3. Не сравнивайте логическую переменную со значениями == *истина* или == *ложь*

- В этом сравнении нет необходимости, поскольку логическая переменная сама по себе является логическим выражением.

```
# Bad example
if is_cold == True:
    print("The weather is cold!")
```

3. Не сравнивайте логическую переменную со значениями == *истина* или == *ложь*

- Это допустимое условие, но мы можем сделать код более читабельным, используя вместо него саму переменную как условие, как показано ниже.

```
# Good example
if is_cold:
    print("The weather is cold!")
```

- Если вы хотите проверить, является ли логическое значение истинным, вы можете использовать оператор not.

Проверка истинного значения

- Если мы используем нелогический объект в качестве условия в операторе **if** вместо логического выражения, Python проверит его истинное значение и использует его, чтобы определиться, следует ли выполнить код с отступом. По умолчанию значение истинности объекта в Python считается истинным, если не указано как ложное в документации.
- Вот большинство встроенных объектов, которые считаются ложными в Python
  - константы, определенные как ложные: **None** и **False**
  - нуль любого числового типа: 0, 0.0, 0j, десятичное число(0), дробь(0,1)
  - Пустые последовательности и коллекции: ‘ ””, (), [], {}, набор(), диапазон (0)

Пример:

```
errors = 3
if errors:
    print("You have {} errors to fix!".format(errors))
else:
    print("No errors to fix!")
```

- В этом коде **ошибки** имеют истинное значение, поскольку это не ненулевое число, поэтому печатается сообщение об ошибке. Это хороший, краткий способ написания оператор **if**.

Тест: Булевы выражения для условий

1. Даны два целых числа. Выведите значение наименьшего из них
2. В математике функция sign(x) (знак числа) определена так:

sign(x) = 1, если x > 0,

sign(x) = -1, если x < 0,

sign(x) = 0, если x = 0.

Для данного числа x выведите значение sign(x). Эту задачу желательно решить с использованием каскадных инструкций if... elif... else.

3. Даны три целых числа. Выведите значение наибольшего из них.

Циклы For

- В Python есть два типа цикла – циклы **for** и **while**. Цикл **for** используется для «итерации» или повторного выполнения чего-либо над **итерируемым.**
- **Итерируемый -** это объект, который может возвращать один из своих элементов за один раз. Это может включать типы последовательности, такие как строки, списки и кортежи, а также непоследовательные типы, такие как словари и файлы.
- **Пример:**
- Давайте разберем компоненты цикла **for**, используя этот пример со списком **городов:**

```
cities = ['new york city', 'mountain view', 'chicago', 'los angeles']
for city in cities:
    print(city)
print("Done!")
```

Компоненты цикла *for*

- Первая строка цикла начинается с ключевого слова **for**, которое обозначает, что это цикл **for**
- После этого указывается **город в городах**, указывая, что **город -** это итерационная переменная, а **города** - это итерируемый объект, по которому проходят. В первой итерации цикла **город** получает значение первого элемента в **городах**, а именно, «город Нью-Йорк».
- Строка заголовка цикла **for** всегда заканчивается двоеточием**:**
- После заголовка цикла for следует блок кода с отступом, тело цикла, которое выполняется в каждой итерации этого цикла. В теле этого цикла есть только одна строка – **печать (город).**

Компоненты цикла *for*

- После выполнения тела цикла мы еще не переходим к следующей строке; мы возвращаемся к строке заголовка **for**, где переменная итерации принимает значение следующего элемента итерируемого. Во второй итерации приведенного выше цикла, **город** принимает значение следующего элемента в **городах**, а именно «вид на горы».
- Этот процесс повторяется до тех пор, пока цикл не пройдет через все элементы итерируемого. Затем мы переходим к строке, которая следует за телом цикла - в этом случае **печатается (Сделано!).** Мы можем сказать, какая следующая строка после тела цикла, потому что у нее нет отступа. Вот еще одна причина, почему в Python очень важно обращать внимание на ваши отступы!

Компоненты цикла *for*

- Выполнение кода в приведенном выше примере дает следующий результат:

```
new york city
mountain view
chicago
los angeles
Done!
```

- Вы можете именовать итерационные переменные так, как вам нравится. Распространенным шаблоном является присвоение переменной итерации и итерируемому одинаковых имен, за исключением версий в единственном и множественном числе соответственно (например, «город» и «города»).

Использование функции *range ()* с циклами *for*

- **range () - это** встроенная функция, используемая для создания итерируемой последовательности чисел. Вы будете часто использовать **range ()** с циклом **for,** чтобы повторять действие определенное количество раз. Любая переменная может использоваться для обхода чисел, но программисты Python традиционно

```
for i in range(3):
    print("Hello!")
```

- Результат:

```
Hello!
Hello!
Hello!
```

range(start=0, stop, step=1)

- Функция **range ()** принимает три целочисленных аргументов, первый и третий из которых являются дополнительными:
  - Аргумент 'start' является первым номером последовательности. Если не указывается, 'start' принимает значение 0 по умолчанию.
  - Аргумент 'stop' на 1 больше, чем последний номер последовательности. Этот аргумент должен быть указан.
  - Аргумент 'step' разница между каждым числом в последовательности. Если не указывается, 'step' принимает значение 1 по умолчанию.
- Примечания по использованию функции **range ()**:
  - Если вы указываете одно целое число в скобках с помощью **range ()**,оно используется в качестве значения для 'stop,' а значения по умолчанию используются для двух других.
    - **например – range(4)** возвращает **0, 1, 2, 3**
  - Если вы указываете два целых числа в скобках с помощью (),они используются для 'start' и 'stop,' а значение по умолчанию используется для 'step.`
    - **например – range(2, 6)** возвращает **2, 3, 4, 5**
  - Или вы можете указать все три целых числа для 'start', 'stop' и 'step.`
    - **например – range(1, 10, 2)** возвращает **1, 3, 5, 7, 9**