Урок 3.  
Условный оператор. Вложенность.

короткая линия

# План урока

1. Условная конструкция
2. Вложенность
3. Ключевое слово elif
4. Логические операторы
5. Дополнительно: тернарный оператор.

# 

# Условная конструкция

На прошлом уроке мы проходили сравнения чисел и строк, но почти их не использовали. По-настоящему эффективно сравнения применяются в связке с условными конструкциями. Условные конструкции позволяют делать ваш код более гибким и “послушным”, направляя команды программе в зависимости от того, что она получает, или того, как меняются переменные в ходе ее выполнения. Ключевыми словами в подобной команде являются if (“если”) и else (“иначе”). Используется она так:

|  |
| --- |
| **if** condition:  statements1 **else**:  statements2 |

где condition – это проверяемое условие, после которого идет двоеточие, означающее определение нового блока кода (заметим, что условие не обязательно писать в круглых скобках, как в некоторых других языках), а statements1 — команды, которые выполняются при соблюдении данного условия. Если же условие оказалось неверным, выполняются команды statements 2, определенные в блоке else.

Пользуясь данным примером, решите следующие задачи:

**Задача “Чет-нечет”**

**Задача “Максимум и минимум без функций”**

**Задача “Большое число”**

**Задача “Равные строки”**

**Задача “Две операции арифметики”**

# Вложенность

В языке Python, чтобы выделить команды, относящиеся к определенному блоку, используют отступы. Обычно разработчики используют 4 пробела или один знак табуляции как наиболее визуально удобный подход, но технически вы можете использовать любое количество пробелов (главное, одинаковое количество на программу). В других языках программирования (например, C/C++, Java, PHP, JavaScript) такое также применяется, однако вместо двоеточия команды, относящиеся к блоку, заключаются в фигурные скобки, а вот в языке Pascal используются ключевые слова begin и end.

В свою очередь, в одном блоке может содержаться бесчисленное множество блоков, в каждом из которых могут содержатся собственные блоки программного кода. Схематично это выглядит так:

|  |
| --- |
| **if** condition1:  **if** condition2:  **if** condition3:  statements3  statements2  statements1 |

Как видите, отступы делают программу удобочитаемой. Это всегда очень важно, ведь к своему коду вы можете возвращаться не раз, а исправить баг или добавить новую “фичу” в неряшливый код бывает довольно трудно. То же касается и ситуаций, когда вы работаете в команде, и ваш код будут просматривать ваши коллеги.

**Задача “Длина числа”**

**Задача “Возрастающая последовательность”**

# Ключевое слово elif

Мы уже говорили о том, что Python довольно лаконичный язык. Действительно, многие вещи в нем можно написать гораздо проще, чем в других языках. Рассмотрим пример. Допустим, нам нужно проанализировать введенное число, сравнив его с числом десять.

|  |
| --- |
| n = int(input()) **if** n < 10:  print(**"Число меньше десяти"**) **else**:  **if** n == 10:  print(**"Число равно десяти"**)  **else**:  **if** n > 100:  print(**"Число намного больше десяти"**)  **else**:  print(**"Число больше десяти"**) |

В итоге, для такой простой проверки понадобилось несколько вложенных друг в друга кусков кода, которые могут выглядеть довольно отталкивающе. К тому же, если бы нам понадобилось больше, чем три-четыре различные ситуации, код стал бы похож на лесенку, уходящую в бесконечность. Любой Python разработчик сказал бы, что этот код выглядит ужасно. Попробуем улучшить его с помощью команды elif. По названию можно догадаться, что elif соединяет в себе слова else и if и делает код более кратким. Посмотрим, что у нас получилось:

|  |
| --- |
| n = int(input()) **if** n < 10:  print(**"Число меньше десяти"**) **elif** n == 10:  print(**"Число равно десяти"**) **elif** n > 100:  print(**"Число намного больше десяти"**) **else**:  print(**"Число больше десяти"**) |

Думаю, вы согласитесь, что код теперь выглядит гораздо понятнее и красивее.

**Задача “Манхэттенский проект”**

**Задача “Точки в системе координат”**

**Задача “Калькулятор”**

# Логические операторы

Часто бывает, что необходимо проверить больше одного условия за раз. Чтобы избежать лишней вложенности, используют логические операторы: and, or и not. Рассмотрим использование каждого из них на примерах:

1. Написать “YES”, если длина строки text равна 42 **или** строка равна строке “Python”.

|  |
| --- |
| **if** len(text) == 42 **or** text == **"Python"**:  print(**"YES"**) |

2. Написать “YES”, если число n шестизначное **и** при этом последняя цифра равна 2.

|  |
| --- |
| **if** 10 \*\* 5 <= n < 10 \*\* 6 **and** n % 10 == 2:  print(**"YES"**) |

3. Написать “right\_choice”, если **неверно, что** строка language\_to\_choose равна “PHP” или равна “JavaScript”. Обратите внимание на скобки в примере. Это значит, что отрицание берется на все выражение в скобках, идущее после слова not.

|  |
| --- |
| **if** **not** (language\_to\_choose == **"PHP"** **or** language\_to\_choose == **"JavaScript"**):  print(**"right\_choice"**) |

Очевидно, с помощью комбинаций логических операторов (не забывайте про скобки) можно писать сколь угодно сложные условия.

**Задача “Високосный ли?”**

**Творческая задача “Искусственный интеллект: Начало”**

Стоит упомянуть и так называемые “операторы членства”, с помощью которых можно проверить вхождение произвольной подстроки в произвольную строку. Например, код

|  |
| --- |
| **if** **"Париж"** **in** **"Города, в которых я был: Москва, Мельбурн, Квебек, Мадрид, Кейптаун, Париж, Флоренция"**:  print(**"Content d'entendre!"**) |

выведет “Content d'entendre!”, так как “Париж” действительно присутствует в строке. Если же вы хотите удостовериться в **отсутствии** подстроки, используйте оператор **not in**.

**Задача “Валидация пароля. Этап 1.”**

**Задача “Резюме каучсерфера”**

**Задача “Сознательный шоппинг”**

# Тернарный оператор

В Python также существует возможность еще более сократить проверку условия — буквально в одну строку. Делается это так:

|  |
| --- |
| a = int(input()) b = int(input()) res = **"Число а больше"** **if** a > b **else** **"Число b больше"** |

В общем виде тернарный оператор выглядит так:

|  |
| --- |
| x = value1 **if** condition **else** value2 |

где value1 и value2 — это переменные, которые “кладутся” в x при выполнении или невыполнении условия. Заметим, что в тернарном операторе нет места elif, так что использовать его стоит в однозначных случаях. В нашем примере, если числа равны, в переменную res сохранится строка, содержащая фактологическую ошибку.

**Домашнее задание:**

**Задача “Ты не пройдешь!!”**

**Задача “Шахматная партия питонистов”**

**Творческая задача “Искусственный интеллект: Возвышение”**

**Творческая задача “Skyrim: Very Special Edition”**