Урок 5.  
Цикл while и логические переменные.

короткая линия

# План урока

1. Логические переменные
2. Цикл while
3. Операторы break и continue
4. “Вспомнить все”: изучение алгоритмов

# Логические переменные

Продолжим наше изучение типов данных в Python (на данный моменты мы знакомы с целыми и вещественными числами, строками и кортежами) и коснемся темы логических переменных. Логическими, или булевыми (в честь математика [Джорджа Буля](https://ru.wikipedia.org/wiki/%D0%91%D1%83%D0%BB%D1%8C,_%D0%94%D0%B6%D0%BE%D1%80%D0%B4%D0%B6)), называют переменные, которые могут принимать лишь два значения – 1 или 0, которые записываются как True или False соответственно. Это удобно, так как занимает меньше памяти, а также в таком представлении удобно хранить **истинность или ложность** какого-либо утверждения. Например, вы разрабатываете онлайн-игру и вам необходимо отображать, находится ли игрок в сети в данный момент времени. Вы можете завести логическую переменную is\_online и менять ее на 1 в тот момент, когда игрок входит игру, и на 0, когда он выходит.

Если вы вспомните про операторы сравнения, вы поймете, что они возвращают как раз логическое выражение. Вы можете убедится в этом, определив переменные a и b и написав print(a == b).

**Задача “Pied Piper”**

# Цикл while

Напомним, что мы использовали цикл for, когда точно знали, сколько раз нужно повторить код, относящийся к циклу. Однако цикл while позволяет нам производить итерации, **пока** выполняется заданное нами условие. Образец использования в общем виде выглядит так:

|  |
| --- |
| **while** условие:  блок кода |

При выполнении цикла while сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается, а если истинно, то выполняется блок кода, после чего условие проверяется снова. Так продолжается до тех пор, пока условие будет истинно.

Посчитаем количество цифр числа с помощью цикла while:

|  |
| --- |
| n = int(input()) length = 0 **while** n > 0:  n //= 10  length += 1 print(**"Длина числа -"**, length) |

**Задача “Валидация пароля. Этап 2.”**

**Задача “Спонтанная сумма”**

**Задача “Вдохновение писателя. Этап 1.”**

**Задача “Учимся ~~экономить~~ транжирить”**

# break, continue и else

Для управления кодом в цикле while используются операторы break и continue. Как только интерпретатор доходит до break в цикле while, происходит немедленный выход из цикла. break можно помещать, например, в условие внутри цикла.

|  |
| --- |
| languages = (**"Java"**, **"Python"**, **"C++"**, **"PHP"**, **"Pascal"**, **"Ruby"**) i = 0 **while** i < len(languages):  **if** languages[i] == **"PHP"**:  print(**"NO!"**)  **break**  print(**"Cool!"**)  i += 1 |

Оператор continue немедленно начинает следующую итерацию цикла, минуя оставшийся код в блоке.

|  |
| --- |
| **for** i **in** range(10):  print(**'Загружаемся... Ждите.'**)  **if** i == 3:  print(**'Упс... Ошибочка вышла. Делаем вид, что ничего не было.'**)  **continue**  print(**'Загружено'**, i \* 10, **'процентов'**) print(**'Загрузка завершена!'**) |

else, в свою очередь, используется в связке с break. Код, помещенный в else, **НЕ** выполняется, если выход из цикла произошел из-за break.

Данные операторы применяются достаточно редко из-за того, что их почти всегда можно заменить грамотным выставлением условия, но иногда их использование все же бывает оправданным.

**Задача “The Hunt of Pied Piper”**

# Алгоритмы

На данный момент знание алгоритмов для многих программистов не является решающим фактором, определяющим возможность/невозможность успешного выполнения той или иной задачи. Причин этому несколько: во-первых, множество библиотек (обширного набора инструментов – мы будем проходить это во втором полугодии) обеспечивают быстрый доступ к основным функциям и возможностям, поэтому многие решают не изобретать велосипед и “не забивать голову” тем, чем они, возможно, никогда не воспользуются, и во-вторых, многие разработчики вообще не сталкиваются с ситуациями, где необходимы алгоритмы (в их число входят фронтендеры, администраторы баз данных, программисты 1С). Отсюда же вытекает спор “что же лучше?”, касающийся промышленного (разработка приложений для пользователя) и олимпиадного программирования. Очевидно, что на данный вопрос нельзя ответить однозначно. Учить или не учить алгоритмы – решать вам. Мы лишь коснемся нескольких фундаментальных алгоритмов, которые знать необходимо. Выучив алгоритм, вы сможете реализовать его на любом языке программирования, зная его синтаксис.

Почитать о разных точках зрения на вопрос можно тут:

<https://habr.com/post/279453/>

<https://habr.com/post/279651/>

Ниже представлена шпаргалка основных алгоритмов:

**Нахождение максимума**

|  |
| --- |
| numbers = (1, 2, 3, ..., n) *# кортеж чисел* max\_number = numbers[0] i = 1 **while** i < len(numbers):  **if** numbers[i] > max\_number:  max\_number = numbers[i]  i += 1  print(max\_number) |

**Нахождение минимума**

|  |
| --- |
| numbers = (1, 2, 3, ..., n) *# кортеж чисел* min\_number = numbers[0] i = 1 **while** i < len(numbers):  **if** numbers[i] < max\_number:  min\_number = numbers[i] print(min\_number) |

**Нахождение суммы**

|  |
| --- |
| numbers = (1, 2, 3, ..., n) *# кортеж чисел* sum = 0 **for** number **in** numbers:  sum += number print(sum) |

**Проверка на простоту**

d = 2  
**while** d \* d <= n **and** n % d != 0:  
 d += 1  
print(d \* d > n)

**Нахождение НОДа (алгоритм Евклида)**

|  |
| --- |
| a, b = int(input()), int(input()) **while** b != 0:  a, b = b, a % b print(a) |

**Домашнее задание**

**Задача “Минимальный делитель”**

**Задача “Построение простых”**

**Задача “Степень ли?”**

**Задача “НОД нескольких”**

**Задача “Olympiadnik Starter Pack”**