Урок №2. Условный оператор, отступы, числа, функции

План урока

1. Повторение.
2. Условный оператор (`if`, `else`, позже `elif`), блоки кода и отступы.
3. Типы данных, целые и вещественные числа, арифметика.
4. Функции, функция `len`, преобразование типов.

Аннотация

Во втором уроке впервые рассматривается несколько фундаментально важных тем: условный оператор, типы данных и функции.

Повторение

Задача «Вспомнить всё» (https://lms.yandexlyceum.ru/task/view/193).

Необязательная задача «Эхо-0» + (https://lms.yandexlyceum.ru/task/view/212).

Сегодня мы изучим условный оператор, научимся работать с числами и поговорим о функциях.

Условный оператор используется, когда некая часть программы должна быть выполнена, только **если** верно какое-либо условие. Для записи условного оператора используются ключевые слова `if` и `else` («если», «иначе»), двоеточие, а также **отступ в четыре пробела**.

Обратите внимание: в начале условного оператора `if` выполняется сравнение, а не присваивание. Разница вот в чём: **сравнение** — это проверка, которая не меняет значение переменной (в сравнении может вообще не быть переменных), а присваивание — команда, которая меняет значение переменной. Для сравнения нужно использовать двойной знак равенства: ==. После else никогда не пишется никакого условия.

```
print('Введите пароль:')
password = input()
if password == 'qwerty':
    print('Доступ открыт.')
else:
    print('Ошибка, доступ закрыт!')
```

Другой пример:

```
print('Представься, о незнакомец!')
name = input()
if name == 'Цезарь':
    print('Аве, Цезарь!')
else:
    print('Приветик.')
```

Задача «Только питон!» (https://lms.yandexlyceum.ru/task/view/194)

Необязательная задача «Дзен» (https://lms.yandexlyceum.ru/task/view/213).

Иногда в условном операторе нужно задать сложное условие. Для этого можно использовать **логические операции** and («и»), or («или») и not («не»), а также сравнение != («не равно»). Например, вот так можно проверить, что оба условия выполнены:

```
print('Как называются первая и последняя буквы греческого алфавита?')
greek_letter_1 = input()
greek_letter_2 = input()
if greek_letter_1 == 'альфа' and greek_letter_2 == 'омега':
    print('Верно.')
else:
    print('Неверно.')
```

Ниже еще несколько примеров.

```
print('Как греки или римляне называли главу своего пантеона - бога грома?')
ancient_god = input()
if ancient_god == 'Зевс' or ancient_god == 'Юпитер':
    print('Верно.')
else:
    print('Неверно.')
print('Введите имена двух братьев из античных мифов и легенд.')
brother1 = input()
brother2 = input()
if brother1 == 'Ромул' and brother2 == 'Рем' or brother1 == 'Кастор' and (brother2 ==
'Поллукс' or brother2 == 'Полидевк'):
    print('Верно.')
else:
    print('Неверно.')
print('Введите любые два слова, но это не должны быть "белый" и "медведь" разом.')
word1 = input()
word2 = input()
if not (word1 == 'белый' and word2 == 'медведь'):
    print('Верно.')
else:
    print('Неверно.')
```

Если в арифметических формулах сначала выполняется умножение, а потом сложение, то в логических выражениях сначала выполняется «не» (not), затем «и» (and), затем «или» (or)

Задачи

«Ёлочка, гори» (https://lms.yandexlyceum.ru/task/view/195).

«Ёлочка-2» (https://lms.yandexlyceum.ru/task/view/196).

«Ёлочка-3» (https://lms.yandexlyceum.ru/task/view/197).

Необязательные задачи

«Да или нет?! (https://lms.yandexlyceum.ru/task/view/214)».

«Каникулы капризного ребёнка (https://lms.yandexlyceum.ru/task/view/216)».

В команде `if` при выполнении условия можно выполнять более одной команды. Для этого все их необходимо выделить отступом. Такая запись называется **блоком кода**. По отступам интерпретатор определяет, какие команды исполнять при выполнении каких условий. Аналогично можно делать и для команды `else`.

```
print('Представься, о незнакомец!')
name = input()
if name == 'Цезарь' or name == 'Caesar':
    print('Аве, Цезарь!')
    print('Слава императору!')
else:
    print('Приветик.')
    print('Погода сегодня хорошая.')
print('Засим - заканчиваем.')
```

Перед последней строчкой нет отступа — это означает, что она будет выполнена в конце работы программы в любом случае. А вот две предыдущие строчки будут выполнены, только если условие `if` окажется ложным.

Блоки кода в Питоне очень гибко устроены: внутри них можно писать любой другой код, в том числе условные операторы. Среди команд, которые выполняются, если условие `if` истинно («внутри if») либо ложно («внутри else»), могут быть и другие условные операторы. Тогда команды, которые выполняются внутри этого внутреннего `if` или `else`, записываются с дополнительным отступом.

Изучите пример ниже. `elif` — это короткая запись для "else: if". Если не пользоваться короткой записью, то `if` . пришлось бы писать на отдельной строчке и с отступом (а всё, что внутри этого `if` — с дополнительным отступом). Это не очень удобно, и `elif` избавляет от такой необходимости.

```
print('Представься, о незнакомец!')
name = input()
if name == 'Цезарь' or name == 'Caesar':
    print('Аве, Цезарь!')
    print('В честь какого бога устроим сегодня празднество?')
    god = input()
    if god == 'Юпитер':
        print('Ура Громовержцу!')
    elif god == 'Минерва': # если оказалось, что имя бога не 'Юпитер', то проверяем, на равно ли оно строке 'Минерва'
        print('Ура мудрой воительнице!')
    else: # следующая строка будет выполнена, только если имя бога не 'Юпитер' и не 'Минерва'
        print('Бога по имени', god, 'мы не знаем, но слово Цезаря - закон.')
    print('Слава императору!') # эта команда будет выполнена независимо от того, какое имя бога ввёл пользователь, если только изначально он представился Цезарем
else:
    print('Приветик.')
    print('Погода сегодня хорошая.')
print('Засим - заканчиваем.')
```

Задача «Личностный тест» (https://lms.yandexlyceum.ru/task/view/204) + .

Необязательная задача «Личностный тест 2» (https://lms.yandexlyceum.ru/task/view/215) + .

Теперь рассмотрим новую команду для работы со строками — команду in. Эта команда позволяет проверить, что одна строка находится внутри другой (например, строка «на» находится внутри строки «сложная задача»). В таком случае обычно говорят, что одна строка является **подстрокой** для другой.

```
text = input()
if 'хорош' in text and 'плох' not in text:
    print('Текст имеет положительную эмоциональную окраску.')
elif 'плох' in text and 'хорош' not in text:
    print('Текст имеет отрицательную эмоциональную окраску.')
else:
    print('Текст имеет нейтральную или смешанную эмоциональную окраску.')
```

Первое условие окажется истинным, например, для строк «всё хорошо» и «какой хороший день», но не для «ВсЁ ХоРоШо» и не для «что-то хорошо, а что-то и плохо». Аналогично, второе условие окажется истинным для строк «всё плохо», «плохое настроение» и т. д.

Задача «Мяу» (https://lms.yandexlyceum.ru/task/view/198)

Необязательная задача «Регистрация почты» (https://lms.yandexlyceum.ru/task/view/217)

Пока единственным **типом данных**, с которым мы работали, были **строки**. Теперь рассмотрим **целые и вещественные числа**. У каждого элемента данных, который встречается в программе, есть свой тип. (В случае Питона более правильный термин — «класс объекта», но об этом мы будем говорить гораздо позже.) Например, 'привет' — это строка, а вот 15.3 — это число (дробное). Даже если данные не записаны прямо в программе, а получаются откуда-то ещё, у них есть совершенно определённый тип. Например, на место input() всегда подставляется строка, а 2+2 даст именно число 4, а не строку '4'.

В Питоне есть два типа чисел: целые и вещественные. В качестве разделителя целой и дробной части вещественных чисел используется точка (не запятая!). С числами можно выполнять разные арифметические операции: сложение (+), вычитание (-), умножение (*), деление (/), деление нацело (//), остаток от деления (%), возведение в степень (**).

```
a = 7
b = 2
z = a + b
print(z, a-b, a*b, a/b, a//b, a%2, a**b)
# 9 5 14 3.5 3 1 49
if a >= b > 0:
    print(a, 'больше либо равно', b)
    print(a, 'и', b, '- положительные числа.')
```

Задача «Количество минут в году» (https://lms.yandexlyceum.ru/task/view/205).

Обратите внимание, что '1' и 1 — совершенно разные вещи. Интерпретатор Питона считает строку просто набором символов. Ему неважно, что некоторыми из этих символов мы обозначаем числа. С другой стороны, ему неважно, что число можно записать как последовательность цифр. И строку, и число можно положить в переменную, вывести на экран и сравнить с другой сущностью того же типа (строкой или числом).

Однако прочие возможности у них совершенно разные. В строке можно искать подстроку (`in`), а в числе — нельзя. Числа можно умножать и возводить в степень, а строки — нельзя. Зато строки можно сравнивать (алфавитный порядок). Их можно и складывать, но иначе, чем числа: 'абв'+'где' даст 'абвгде', '1'+'2' даст '12', в то время как 1+2 даст, конечно, 3.

Задача «Факториал: первое знакомство» (https://lms.yandexlyceum.ru/task/view/199).

Задача «Уравнение» (https://lms.yandexlyceum.ru/task/view/206). +

Важное новое понятие — **функция**. В математике функция из одного числа (или даже нескольких) делает другое. Так и в программировании (и в Питоне в частности): функция — это сущность, которая из одного (или даже нескольких) значений делает другое. При этом она может ещё и выполнять какие-то действия. Выглядит это так:

```
word = input()
length = len(word)
print('Вы ввели слово длиной', length, 'букв.')
```

Попробуйте запустить эту программу.

Здесь используется стандартная функция Питона `len`, которая вычисляет длину строки. Когда мы в программе используем функцию, это называется «**вызов функции**». Вызов функции устроен так: пишем имя функции — `len`, а затем в скобках те данные, которые мы передаём этой функции, чтобы она что-то с ними сделала. Такие данные называются **аргументами**.

В нашем случае данные в скобках должны быть строкой. Мы выбрали в качестве данных значение переменной `word`, которое пользователь до этого ввёл с клавиатуры. То есть значение переменной `word` выступает здесь в роли аргумента. А функция `len` выдаёт длину этой строки. Если пользователь ввёл, например, «привет», то `word` оказывается равно «привет», и на место `len(word)` подставляется длина строки «привет», то есть 6.

Обратите внимание: каждый раз, когда мы пишем имя переменной (кроме самого первого раза — в операторе присваивания слева от знака =), вместо этого имени интерпретатор подставляет значение переменной. Точно так же на место вызова функции (то есть имени функции и её аргументов в скобках) подставляется результат её работы — это называется **возвращаемое значение** функции. Таким образом, функция `len` возвращает длину своего аргумента. Вообще-то функция сама по себе — это фактически небольшая программа, но об этом позже.

Функции безразлично происхождение значений, которые ей передали в качестве аргумента. Это может быть значение переменной, результат работы другой функции или записанное прямо в коде значение:

```
length = len('абракадабра')
print('Это слово длиной', length, 'букв.')
```

Обратите внимание, что в предыдущем примере значение переменной word вообще никак не изменилось от вызова функции `len`. С другой стороны, вызов функции может стоять где угодно, не обязательно сразу класть возвращаемое значение в переменную:

```
word = input()
print('Вы ввели слово длиной', len(word), 'букв.')
```

Иногда сама функция делает что-то ещё. `input()` — тоже функция (отсюда скобки), она не принимает никаких аргументов, зато считывает строку с клавиатуры и возвращает её. И `print` — тоже функция, она не возвращает никакого осмысленного значения, зато выводит свои аргументы на экран. Эта функция может принимать не один аргумент, а сколько угодно. Несколько аргументов одной функции следует разделять запятыми.

Задача (для разбора). Предположим, у нас имеется программа (странноватая, не всегда логичная) — часть интерфейса программы для дизайна визитных карточек. Как будет работать эта программа?

```
max_length = len('Константин Константинович Константинопольский')
name = input()
print('Здравствуйте,', name)
second_name = input()
surname = input()
full_name = name + ' ' + second_name + ' ' + surname
full_name_length = len(name) + len(second_name) + len(surname) + 2
if len(full_name) != full_name_length:
    print('Ошибка: что-то пошло не так.')
else:
    if len(full_name) > max_length:
        print('Извините, ваше имя слишком длинное. Попробуйте изменить имя.')
        name = input()
#       surname = input()
    print(len(name), len(full_name))
print('Спасибо, что воспользовались нашей программой!')
```

Задача «Длина» (https://lms.yandexlyceum.ru/task/view/200)

Необязательная задача «Верстаем визитную карточку (https://lms.yandexlyceum.ru/task/view/218)».

Пока что мы не умеем вводить с клавиатуры числа, ведь функция `input()` всегда возвращает строку. Пользователь может ввести с клавиатуры какие-то цифры, но в результате `input()` вернёт строку, состоящую из этих цифр. Если мы попытаемся, например, прибавить к этой строке 1, то получим ошибку.

Но есть выход — функции! Тип данных «целое число» в Питоне называется `int`, дробное число — `float`. Одноимённые функции принимают в качестве аргумента строку и возвращают число, если в этой строке было записано число (иначе выдают ошибку):

```
number = int('4')
print(number + 1)
```

`int('4')` — это 4, что вовсе не то же самое, что '4'!

Или:

```
number = int(input())
print(number+1)
```

Или даже:

```
number = int(input())
number = number + 1
print(number)
```

Кроме того, если передать функции `int` в качестве аргумента дробное число, то она сделает из него целое путём отбрасывания дробной части. Наконец, строковый тип данных называется `str`, и одноимённая функция сделает из любого числа строку. Все эти манипуляции называют **преобразованием (приведением, конвертацией) типов**.

```
print(int(2.7))
print(str(2) + str(2))
```

Каждый раз, когда вы пишете программу, важно понимать, какой тип имеет каждое значение и каждая переменная.

Задачи

«Сложить два числа (201)»,

«Сложить ещё два числа (202)»,

«Одно число (203)».

«Короткая светская беседа» (207).