**Ввод и вывод данных в Python**

**Получение данных input**

**Синтаксис операции ввода выглядит так**:

a = input()

Если выполнить данный код в интерактивном режиме в IDLE, курсор переместиться новую строку и программа будет ожидать ввода данных, которые будут записаны в переменную a.

Кроме того, оператор input() позволяет перед запросом ввода вывести какой-либо текст, например:

a = input("Введите своё имя")

Таким образом можно дать подсказку пользователю, что нужно ввести. В отличие от строго типизированных языков программирования Python позволяет вводить любые типы данных без их явного указания. То есть пользователь может ввести как [строку](https://all-python.ru/osnovy/stroki.html), так и [число](https://all-python.ru/osnovy/chisla.html). Синтаксис команды ввода от этого не изменится.

Однако, часто нужно получить данные определенного типа, например, калькулятор ожидает число, а пользователь вводит строку. **Для этого используется приведение типа, в данном случае к целочисленному**:

a = int(input("Введите целое число"))

**Вывод данных print**

Для вывода данных в Python используется функция print(). С её помощью можно выводить строки, числа, последовательности. **Её синтаксис выглядит так**:

print("Строка")

print(5)

**В print() можно передать несколько аргументов**:

print("один", "два", 5) # Выведет "один два 5"

**Кроме того, строки можно складывать с помощью оператора “+”**:

print("один" + "два", 5) # Выведет "одиндва 5"

В print() предусмотрены дополнительные параметры. Например, через параметр sep можно указать отличный от пробела разделитель строк:

>>> **print**("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun", sep="-")

Mon-Tue-Wed-Thu-Fri-Sat-Sun

>>> **print**(1, 2, 3, sep="//")

1//2//3

Параметр end позволяет указывать, что делать, после вывода строки. По-умолчанию происходит переход на новую строку. Однако это действие можно отменить, указав любой другой символ или строку:

>>> **print**(10, end="")

10>>>

Обычно, если end используется, то не в интерактивном режиме, а в скриптах, когда несколько выводов подряд надо разделить не переходом на новую строку, а, скажем, запятыми. Сам переход на новую строку обозначается комбинацией символов "\n".

Условные операторы

Оператор if

Условные операторы нужны для проверки условий и, в зависимости от результата, чтобы вести логику выполнения программы в нужном направлении.

Условный оператор **if** ("если") является основным оператором проверки выполнения условия. Для того, чтобы выполнить простую вложенную инструкцию, необходимо проверить условие на соответствие, использовав оператор **if** и прописав после него соответсвующее условие:

 # объявляем переменную  
var = 5  
# выполняем проверку условия  
if var < 10:  
# если условие выполняется, то выполняется вложенная инструкция  
    print("var less than 10")

В данном случае мы объявляем переменную **var** и присваиваем ей значение равное **5**, далее выполняем проверку условия - если переменная меньше **10**, то вывести соответсвующее сообщение.

Оператор **if** производит проверку истинности выражения, т.е. является ли результат выражения логической истиной (**True**) или же ложью (**False**). Далее выполняется вложенная инструкция, если результат выражения является **True**. Если результат выражения является **False**, тогда вложенная инструкция игнорируется.

Оператор else

Оператор **if** позволяет выполнить вложенную инструкцию, если результат выражения **True**, а что, если нам необходимо, чтобы программа могла выполнять действия и в случае **False** результата? Для этого мы можем использовать оператор **else**.Условный оператор **else**("иначе") является продолжением основной инструкции.

# объявляем переменные  
var\_1 = 10  
var\_2 = 9  
# выполняем проверку условия  
if (var\_1 == var\_2):  
# если True  
    print("var\_1 equal var\_2")  
# иначе  
else:  
# выполняется другая вложенная инструкция  
    print("var\_1 not equal var\_2")

Оператор elif

В предыдущем примере мы рассматривали ситуацию, когда нас интересовало всего 2 возможные ситуации - равно, не равно. А что, если для решения определенной задачи нас интересует более двух возможных ситуации? Для этого предусмотрен оператор **elif**(сокращение от конструкции else if), который позволяет добавить дополнительные условия в логику выполнения программы:

# объявляем переменные  
var\_1 = 10  
var\_2 = -10  
# выполняем проверку условия  
if var\_1 == var\_2:  
# если True  
    print("var\_1 equal var\_2")  
# иначе если выполняется другое условие  
elif var\_1 < var\_2:  
# если True для elif  
    print("var\_1 less than var\_2")  
# если False для всех  
else:  
    print("var\_1 more than var\_2")

Количество конструкций **elif** может быть множество, тогда, как **if** и **else** используются по 1 разу в рамках одной инструкции:

if (условие):  
    (выполнение условия)  
elif (другое условие):  
    (выполнение другого условия)  
elif (третье условие):  
    (выполнение третьего условия)  
elif (четвертое условие):  
...  
...   
...   
else:  
    (выполнение при всех других не рассмотренных ранее случаях)

Оператор **if** всегда задает начало новой инструкции.

Цикл for

Цикл **for** менее универсальный, но работает быстрее, чем цикл **while**. Он способен проходить по любому итерируемому объекту, будь то списки, словари, кортежи, строки и не требует ручного увеличения счетчика итераций:

for i in range(1,14):  
    print(i)

Важно помнить, что увеличение счетчика и его начальное значение заложено внутри оператора for и его не надо прописывать отдельно.

Примечание. Если необходимо выводить элементы цикла без перехода на следующую строку, то можно использовать параметр end=' ' для функции print(). Этому параметру можно ничего не передавать, можно передавать пробелы или другие знаки:

for i in range(1,14):  
 print(i, end=' ')  
1 2 3 4 5 6 7 8 9 10 11 12 13

**Функция range**

Как правило, циклы **for** используются либо для повторения какой-либо последовательности действий заданное число раз, либо для изменения значения переменной в цикле от некоторого начального значения до некоторого конечного.

Для повторения цикла некоторое заданное число раз n можно использовать цикл **fo**r вместе с функцией **range**:

**for i in range(n):**

**Тело цикла**

В качестве n может использоваться числовая константа, переменная или произвольное арифметическое выражение (например, 2 \*\* 10). Если значение n равно нулю или отрицательное, то тело цикла не выполнится ни разу.

Если задать цикл таким образом:

**for i in range(a, b):**

**Тело цикла**

то индексная переменная i будеть принимать значения от a до b – 1, то есть первый параметр функции range, вызываемой с двумя параметрами, задает начальное значение индексной переменной, а второй параметр — значение, которая индексная переменная принимать не будет. Если же a≥b, то цикл не будет выполнен ни разу. Например, для того, чтобы просуммировать значения чисел от 1 до n можно воспользоваться следующей программой:

**sum = 0**

**for i in range(1, n + 1):**

**sum += i**

В этом примере переменная i принимает значения 1, 2, …, n, и значение переменной sum последовательно увеличивается на указанные значения.

Наконец, чтобы организовать цикл, в котором индексная переменная будет уменьшаться, необходимо использовать функцию

**range**

 с тремя параметрами. Первый параметр задает начальное значение индексной переменной, второй параметр — значение, до которого будет изменяться индексная переменная (не включая его!), а третий параметр — величину изменения индексной переменной. Например, сделать цикл по всем нечетным числам от 1 до 99 можно при помощи функции

**range(1, 100, 2)**

, а сделать цикл по всем числам от 100 до 1 можно при помощи

**range(100, 0, -1)**

Более формально, цикл

**for i in range(a, b, d)**

 при d > 0 задает значения индексной переменной i = a, i = a + d, i = a + 2 \* d и так для всех значений, для которых i < b. Если же d < 0, то переменная цикла принимает все значения i > b.