**Условные операторы**

[Условный оператор if. Конструкция if-else 1](#_heading=h.30j0zll)

[Практика 1 5](#_heading=h.1fob9te)

[Вложенные условия и множественный выбор. Конструкция if-elif-else 6](#_heading=h.2et92p0)

[Оператор elif 8](#_heading=h.tyjcwt)

[Тернарный условный оператор. Вложенное тернарное условие 10](#_heading=h.3dy6vkm)

[Вложенный тернарный оператор 12](#_heading=h.1t3h5sf)

[Практика 2 13](#_heading=h.4d34og8)

[Логические операторы not, and, or, in, not in в условных операторах 16](#_heading=h.17dp8vu)

[И (and) 16](#_heading=h.3rdcrjn)

[Или (or) 16](#_heading=h.26in1rg)

[Операторы принадлежности 17](#_heading=h.lnxbz9)

[Pattern matching. Конструкция match 17](#_heading=h.35nkun2)

# Условный оператор if. Конструкция if-else

Блочный синтаксис языка Python

Определение 1. Блок — это несколько команд, которые мы считаем за одну большую команду.

В большинстве языков программирования есть различные пометки, обозначающие начало и конец блока. Например, в языке С это фигурные скобки { и }, а в языке Pascal — это слова begin и end. В языке Python никаких специальных знаков начала и конца блока нет. Блок выделяется отступом на четыре пробела вправо.

|  |  |
| --- | --- |
|  |  |

На этом занятии мы познакомимся с работой условного оператора if. Что он делает и для чего нужен? Помните, на прошлом занятии, мы с вами рассматривали операции сравнения, например, проверяли:

4 > 2

или с переменной:

a = 5

a < 0

На выходе получали булевы значения True и False. Но ничего не говорили, как их дальше использовать в программе. Пришло время восполнить этот пробел. Оператор if позволяет выполнить группу операторов, при истинности указанного условия. Давайте я сразу поясню эту конструкцию на конкретном примере. Предположим, мы пишем программу для вычисления модуля числа:

x = -4

**if** x < 0:

    x = -x

**print**(x)

Смотрите, после оператора if записываем условие, то есть, определяем, что x отрицательное число. И если это так, то оператор меньше вернет значение True, условие сработает и будет выполнена строчка x = -x. А вот следующая строчка (функция print()) стоит уже вне этого условия и будет выполняться всегда.

Так как Python «понимает», что находится в блоке условного оператора, а что нет? Все дело в форматировании текста программы. Когда мы ставим двоеточие, то открывается новый блок, где может располагаться множество операторов.

И все конструкции языка Python, что имеют отступ (обычно четыре пробела) относительно оператора if, будут располагаться внутри этого блока.

Именно поэтому строчка x = -x выполняется только при истинности условия x < 0, а функция print() стоит уже после оператора if и выполняется всегда.

Запустим эту программу и убедимся, что все работает. Видим значение 4, то есть, условие x < 0 вернуло значение True, оператор if сработал и выполнилась строчка x = -x.

А давайте теперь определим:

x = 4

В этом случае также увидим 4, но строка x = -x уже не выполнялась, так как условие для значения False не срабатывает. И, смотрите, если функцию print() сместить на уровень предыдущего оператора:

x = 4

**if** x < 0:

    x = -x

**print**(x)

то в консоли мы уже ничего не увидим. Это произошло, как раз, из-за того, что эта функция переместилась внутрь блока оператора if и будет теперь выполняться только при истинности условия. Вернем, снова:

x = -4

и, теперь, видим значение 4 в консоли. Этот пример показывает, что в Python форматирование текста программы имеет ключевое значение.

И благодаря этому, кстати, программы становятся более наглядными и читабельными, а также заставляют начинающих программистов правильно их оформлять.

На мой взгляд – это большой плюс данного языка программирования.

Чтобы все было понятно, приведу еще пару примеров. Предположим, мы хотим поменять значения двух переменных:

a = float(input("a: "))

b = float(input("b: "))

Если a меньше b:

**if** a < b:

    a, b = b, a

**print**(a, b)

Также можно прописывать более сложные, составные условия, например, для проверки попадания числа в заданный диапазон:

x = int(input())

**if** x >= -4 **and** x <= 10:

**print**("x в диапазоне [-4; 10]")

О составных условиях мы с вами уже говорили на отдельном занятии, поэтому здесь все должно быть понятно.

Или, то же самое условие в Python допускается записывать и так:

**if** -4 <= x <= 10:

То есть, в качестве условия можно записывать любые конструкции, которые можно интерпретировать как истину (True) и ложь (False). Даже так, просто указав числовое значение:

x = int(input())

**if** x:

**print**("x = True")

Мы здесь получим истину, если x не равен нулю. Наконец, в условиях можно явно прописывать булево значение:

**if** True:

**print**("True")

Конечно, это лишь пример и практического смысла в таком операторе особо нет.

Наконец, мы можем использовать операторы проверки для списков. Например, студент имеет следующие оценки по результатам сдачи сессии:

marks = [3, 3, 4, 2, 4]

и мы хотим узнать, будет ли он отчислен. Выполним для этого, следующую проверку:

**if** 2 **in** marks:

**print**("студент будет отчислен")

Вот в этом последнем примере нам бы хотелось в противном случае выдать сообщение, что студент успешно сдал сессию. Это лучше всего реализовать с помощью оператора else:

marks = [3, 3, 4, 2, 4]

**if** 2 **in** marks:

**print**("студент будет отчислен")

**else**:

**print**("студент успешно сдал сессию")

То есть, смотрите, если срабатывает первое условие, то блок else не выполняется. И, наоборот, если условие не срабатывает, то выполняется блок else. Это слово можно перевести как «иначе» и получается либо выполнение первого блока, либо второго, но никогда не обоих вместе.

То есть, здесь определены два взаимоисключающих случая. Действительно, студен не может одновременно быть отчисленным и успешно сдавшим сессию (будем полагать, что по собственному желанию он не уходит).

Или другой подобный пример:

x = int(input())

**if** x < 0:

**print**("x - отрицательное число")

**else**:

**print**("x - неотрицательное число")

Очевидно, что x не может быть одновременно отрицательным и неотрицательным, поэтому для проверки такого условия можно использовать оператор else.

В последнем примере этого занятия мы проверим введенное число на четность. Сделать это можно так:

x = int(input())

**if** x % 2 == 0:

**print**("x - четное число")

**else**:

**print**("x - нечетное число")

Надеюсь, из всех этих примеров, вам понятно, что из себя представляет условный оператор if, как в нем можно прописывать различные условия и как работает оператор else.

## Практика 1

**Задание 1.** Вводятся три целых положительных числа в одну строку через пробел. Убедиться, что первые два числа - это катеты прямоугольного треугольника, а третье - его гипотенуза. (Подсказка: проверка делается по теореме Пифагора  ). Если проверка проходит (истинна), то вывести на экран ДА, иначе - НЕТ.

**Входные данные:**

3 4 5

**Выходные данные:**

ДА

**Задание 2.** Вводится четырехзначное число. Проверить, что оно оканчивается на цифру 7. Вывести на экран ДА, если это так и НЕТ - в противном случае.

**Входные данные:**

8117

**Выходные данные:**

ДА

**Задание 3. "Коровы"**

По данному числу n закончите фразу «На лугу пасется...» одним из возможных продолжений: «n коров», «n корова», «n коровы», правильно склоняя слово «корова».

Например: 1 корова, 2 коровы, 5 коров, 125 коров, 414 коров, 424 коровы.

Формат входных данных

Дано целое положительное число n - количество коров

Формат выходных данных

Программа должна вывести правильное(согласованное) окончание слова "корова": коров, корова или коровы, для заданного числа n.

Подсказка-1

Возьмите листок бумаги и выписывайте все согласования:

1 корова

2, 3, 4 коровы

5 коров

n = int(input())  
  
 if n % 10 == 1 and n != 11:  
 print(n, корова)  
 elif 2 <= n % 10 <= 4 and n // 10 != 1:  
 print(n, коровы)  
 else:  
 print(n, коров)

**Задание 4.** "FooBar". Дано целое число.

Если оно делится на 3 без остатка - вывести "Foo",

если делится на 5 - вывести "Bar",

а если делится на 3 и на 5 - вывести "Foobar".

Для всех остальных случаев не выводить ничего.

**Формат входных данных**

Дано целое число.

**Формат выходных данных**

Вывести "Foo", "Bar" или "Foobar"

"Равнобедренный треугольник"

**Задание 5.** Дан треугольник со сторонами a, b и c.

Если треугольник с данными сторона существует, определите, является ли он равнобедренным.

Формат входных данных

Даны три целые положительные числа, длины сторон треугольника.

Формат выходных данных

Вывести "Равнобедренный", "Не равнобедренный" или "Не существует".

Данные для самопроверки

a b c Результат

3 4 5 Не равнобедренный

2 4 19 Не существует

4 6 4 Равнобедренный

4 4 8 Не Существует

4 6 6 Равнобедренный

Подсказки

Треугольник существует только тогда, когда сумма длин любых его двух сторон больше третьей стороны.

# Вложенные условия и множественный выбор. Конструкция if-elif-else

Мы продолжаем изучение условных операторов языка Python. На предыдущем занятии мы познакомились с работой оператора if проверки условий. И говорили, что внутри блока этого оператора могут быть любые конструкции языка Python, в том числе и другие условия. Давайте посмотрим на примерах, как это работает.



Предположим, что сначала в первом условии нам нужно проверить число на четность, а затем, определить, является ли это число цифрой или каким-либо другим числом:

x = int(input())

**if** x % 2 == 0:

**if** 0 <= x <= 9:

**print**("x - четная цифра")

**else**:

**print**("x - четное число")

Как видите, это удобно сделать с помощью вложенного условия. И, обратите внимание, оператор else здесь относится именно ко второму условию, так как записан на одном уровне с ним. А вот, если бы мы его прописали без отступов:

**if** x % 2 == 0:

**if** 0 <= x <= 9:

**print**("x - четная цифра")

**else**:

**print**("x - четное число")

то он стал бы относиться к первому условию. Программа стала бы работать иначе и, вообще говоря, некорректно для данной задачи.

Я верну отступы для блока else и добавлю еще один такой блок для первого условия:

x = int(input())

**if** x % 2 == 0:

**if** 0 <= x <= 9:

**print**("x - четная цифра")

**else**:

**print**("x - четное число")

**else**:

**print**("x - нечетное число")

В результате, у нас обрабатываются все возможные исходы для числа x.

Другой пример – поиск наибольшего среди трех чисел a, b и c:

a = int(input("a: "))

b = int(input("b: "))

c = int(input("c: "))

**if** a > b:

**if** a > c:

**print**("a -> max")

**else**:

**print**("c -> max")

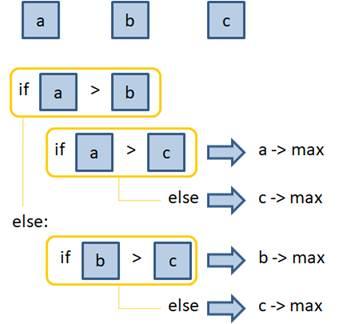
**else**:

**if** b > c:

**print**("b -> max")

**else**:

**print**("c -> max")



Думаю, из этих примеров вам стало понятно, как использовать вложенные условия. Конечно, степень вложенности может быть любой, но все будет работать по аналогии. В практике реального программирования лучше избегать больших вложенностей. Нормальным считается до трех вложений. Если у вас выходит больше, то, скорее всего, что-то делаете не так и структуру программы следует пересмотреть.

## Оператор elif

Во второй части хочу рассказать вам о способе реализации множественного выбора. Что это такое? Представьте, что мы просим пользователя выбрать один из пунктов меню:

1. Курс по Python  
2. Курс по С++  
3. Курс по Java  
4. Курс по JavaScript

Используя текущие знания, мы можем реализовать это так:

item = int(input())

**if** item == 1:

**print**("Выбран курс по Python")

**else**:

**if** item == 2:

**print**("Выбран курс по C++")

**else**:

**if** item == 3:

**print**("Выбран курс по Java")

**else**:

**if** item == 4:

**print**("Выбран курс по JavaScript")

**else**:

**print**("Неверный пункт")

Но эту же логику можно реализовать проще и нагляднее, используя еще один условный оператор

elif = else if

То есть, все конструкции с else if мы можем просто заменить на elif, получим:

**if** item == 1:

**print**("Выбран курс по Python")

**elif** item == 2:

**print**("Выбран курс по C++")

**elif** item == 3:

**print**("Выбран курс по Java")

**elif** item == 4:

**print**("Выбран курс по JavaScript")

**else**:

**print**("Неверный пункт")

Видите, как это удобнее, когда нам в программе нужно реализовать выбор какого-то одного значения из множества доступных. Причем, последовательность операторов должна быть именно такой:

if – elif – else

После if можно опускать любой из них (или оба не прописывать), но порядок следования всегда нужно соблюдать именно таким.

Конечно, помимо проверок на равенство после elif можно прописывать любые условия. Например, вводим с клавиатуры целое положительное x и для него определяем число десятков:

x = int(input())

**if** x < 0:

**print**("x должно быть положительным")

**elif** 0 <= x <= 9:

**print**("x - цифра")

**elif** 10 <= x <= 99:

**print**("x - двузначное число")

**elif** 100 <= x <= 999:

**print**("x - трехзначное число")

В целом это все, что я хотел вам рассказать про условный оператор. Далее мы еще затронем тему тернарных условий. А сейчас вас ждут практические задания по этой теме, после чего переходите к следующему материалу.

## Тернарный условный оператор. Вложенное тернарное условие

На этом занятии по курсу языка Python мы поговорим о тернарном условном операторе, который появился в версии 2.5. Как всегда, вначале поясню, что это за оператор. В простейшем варианте его синтаксис можно представить так:

<значение 1> if <условие> else <значение 2>

Он возвращает значение 1, если условие истинно, а иначе – значение 2. Давайте поясню его работу на конкретном примере. Предположим, у нас имеются две переменные:

a = 12

b = 7

и мы хотим для них определить максимальное значение. Используя имеющиеся знания, это можно было бы сделать так:

**if** a > b:

    res = a

**else**:

    res = b

**print**(res)

А с использованием тернарного оператора это можно реализовать так:

res = a **if** a > b **else** b

Видите, какая простая, понятная и компактная запись в итоге получилась. Это одно из удобств данного оператора. Но, все же, между предыдущей программой с условным оператором if и тернарным оператором есть одно принципиальное отличие. Тернарный оператор автоматически возвращает результат. В данном примере – это или переменная a или переменная b. Тогда как обычный условный оператор предназначен для выполнения блока кода по определенному условию и сам по себе не возвращает никаких значений. То есть, если бы мы попытались записать что-то вроде:

d = **if** a > b:

то попросту возникла бы синтаксическая ошибка – так делать нельзя. А вот результат работы тернарного оператора мы, обычно, сохраняем в некой переменной.

Второе важное отличие – в тернарном операторе нет внутренних блоков, где бы мы могли записывать несколько операторов. Вместо a и b можно прописывать только одну какую-либо конструкцию. Часто это некое значение, или результат работы операторов, например, арифметических:

res = a + 2 **if** a > b **else** b - 5

Так делать вполне допустимо. Или же, можно использовать какие-либо функции:

a = -12

b = -7

res = abs(a) **if** a > b **else** abs(b)

То есть, здесь может быть любая конструкция языка Python, но только одна. По идее, можно даже записать функцию print():

res = **print**(a) **if** a > b **else** **print**(b)

Но тогда переменная res будет принимать значение None, так как функция print() ничего возвращает, то есть, None.

Давайте приведу еще один пример. Пусть имеется некая строка и способ преобразования этой строки:

s = 'python'

type = 'upper'

Пусть 'upper' означает преобразование строки s в верхний регистр. Сделаем это через тернарный оператор, следующим образом:

res = s.upper() **if** type == 'upper' **else** s

Видите, как элементарно и просто реализуется эта идея? В этом большой плюс тернарного оператора. Вообще, его можно рассматривать, как некий активный объект, который возвращает определенное значение в зависимости от условия. Благодаря этому, его можно вызывать прямо внутри различных конструкций, например:

[1, 2, a **if** a < b **else** b, 4, 5]

"a - " + ("четное" **if** a % 2 == 0 **else** "нечетное") + " число"

Обратите внимание здесь на круглые скобки. Они необходимы, чтобы следующий оператор + применялся к результирующей строке, а не к тернарному оператору. Если круглые скобки убрать, то последняя добавка пропадет.

Или его можно указать, как один из аргументов функции:

max(1, 5, a **if** a > 0 **else** b, 4, 5)

Я, надеюсь, из этих примеров вам стал понятен принцип использования тернарного условного оператора.

## Вложенный тернарный оператор

Так как внутри тернарного оператора можно использовать любые конструкции, то кто нам мешает вложить один тернарный оператор в другой, например, так:

<r\_1> **if** <c\_1> **else** <r\_2> **if** <c\_2> **else** <r\_3>

или так:

<r\_1> **if** <c\_1> **else** <r\_2> **if** <c\_2> **else** <r\_3> **if** <c\_3> **else** <r\_4>

Сразу скажу, что это делается крайне редко и лучше избегать таких вложений, так как восприятие и понимание текста программы резко снижается.

Приведу один пример с такими вложениями. На предыдущем занятии мы с вами записывали вложенные условия для определения максимального из трех чисел. Давайте сделаем то же самое, но через тернарные операторы. Программа будет такой:

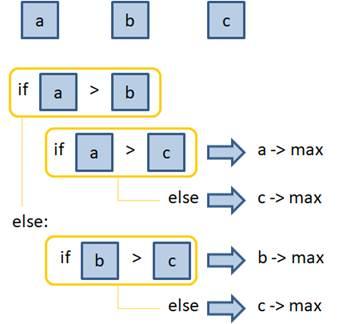
a = 2

b = 3

c = -4

d = (a **if** a > c **else** c) **if** a > b **else** (b **if** b > c **else** c)

**print**(d)



Думаю, общий принцип здесь понятен. Сначала проверяем условие a > b, затем, в зависимости от результата проверки, выполняется или левый тернарный оператор, или правый. Причем, опят же здесь обязательно вложенные тернарные операторы следует взять в круглые скобки, чтобы тернарная конструкция if-else воспринималась как единое целое во вложениях.

Вам осталось закрепить этот материал практическими заданиями и, затем, переходите к следующему материалу.

## Практика 2

**Задание 1. "Симметричное число"**

Дано четырехзначное число. Определите, является ли оно симметричным.

Число считается симметричным, если оно читается одинаково слева направо и справа налево.

Пример симметричного числа: 2442

Формат входных данных

Дано целое положительное четырехзначное число.

Формат выходных данных

Вывести "Да", если число симметричное и "Нет" в противоположном случае.

**Задание 2.** Работа светофора для пешеходов запрограммирована следующим образом: в начале каждого часа в течение трех минут горит зеленый сигнал, затем в течение двух минут – красный, в течение трех минут – опять зеленый и т. д. Дано вещественное число t, означающее время в минутах, прошедшее с начала очередного часа. Определить, сигнал какого цвета горит для пешеходов в этот момент. На экран вывести сообщение (без кавычек) "green" - для зеленого и "red" - для красного.

**Входные данные:**

12.5

**Выходные данные:**

green

**Задание 3.** Вводится два вещественных числа, каждое с новой строки. Необходимо с помощью тернарного условного оператора наибольшее значение присвоить переменной d и вывести ее на экран.

**Входные данные:**

5.4

-3.8

**Выходные данные:**

5.4

**Задание 4.** Вводится вес боксера-любителя (в кг, в виде вещественного числа). Известно, что вес таков, что боксер может быть отнесен к одной из весовых категорий:

1) легкий вес – до 60 кг (включительно);

2) первый полусредний вес – до 64 кг (включительно);

3) полусредний вес – до 69 кг (включительно);

4) остальные - более 69 кг.

Вывести на экран номер категории, в которой будет выступать боксер.

**Входные данные:**

62.4

**Выходные данные:**

2

**Задание 5(ДЗ).** Вводится порядковый номер месяца (1, 2, ..., 12). Вывести на экран количество дней в этом месяце. Принять, что год не является високосным. Реализовать через условный оператор, в котором следует использовать не более трех ветвей (блоков).

P.S. Число дней в месяцах не високосного года, начиная с января: 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31

**Входные данные:**

2

**Выходные данные:**

28

**Задание 6.** Вводится слово. Проверить, что в этом слове присутствуют все три буквы: t, h и o (в произвольном порядке). Реализовать программу с помощью одного условного оператора. Если проверка проходит, вывести ДА, иначе - НЕТ.

**Входные данные:**

Python

**Выходные данные:**

ДА

**Задание 7.** Вводится целое число 0 или 1. Необходимо преобразовать их в строки: 0 - в "False", 1 - в "True". Реализовать это с помощью тернарного условного оператора. Результат отобразить на экране.

**Входные данные:**

1

**Выходные данные:**

True

**Задание 8.** Вводится слово. Необходимо определить, является ли это слово палиндромом (одинаково читается вперед и назад, например, АННА). Регистр букв не учитывать. Если введенное слово палиндром, на экран вывести ДА, иначе - НЕТ.

**Входные данные:**

Шалаш

**Выходные данные:**

ДА

**Задание 9.** Вводится шестизначное число. Определить, является ли оно счастливым. (Счастливым называют такое шестизначное число, в котором сумма его первых трех цифр равна сумме его последних трех цифр.). Вывести ДА, если счастливое и НЕТ - в противном случае.

**Входные данные:**

811235

**Выходные данные:**

ДА

**Задание 10.** "Точка в круге"

Даны координаты точки, координаты центра круга и его радиус.

Определить принадлежит ли данная точка кругу.

Будем считать, что точка принадлежит кругу, если находится внутри его или на его окружности

Формат входных данных

Даны координаты точки (x, y) - два целых числа.

Координаты центра круга (xr, yr) - два целых числа.

Радиус круга - целое положительное число.

Формат выходных данных

Вывести "Да", если точка принадлежит кругу, и "Нет" в противоположном случае.

**import** math

x = float(input("Введите координату x: "))

y = float(input("Введите координату y: "))

r = float(input("Радиус круга r: "))

h = math.sqrt(x \*\* 2 + y \*\* 2)

**if** h <= r:

**print**("Да")

**else**:

# Логические операторы not, and, or, in, not in в условных операторах

Это союзы, которые позволяют объединять по несколько условий. В Python есть всего три оператора: and (и), or (или) и not (не).

## И (and)

Если условия с двух сторон оператора and истинны, тогда все выражение целиком считается истинным.  
**Пример:**

>>> a = 7 > 7 and 2 > -1

>>> print(a)

False

**Пример:**

PetrovSkills = **'Python, Java, Javascript, C++, php'  
if 'Python' in** PetrovSkills **and 'Javascript' in** PetrovSkills:  
 print(**'кандидат на fullstack'**)  
**else**:  
 print(**'в лист ожидания'**)

## Или (or)

Выражение ложно, если оба операнда с двух сторон ложные. Если хотя бы одно из них истинное, то и все выражение истинно.

**Пример:**

>>> a = 7 > 7 or 2 > -1

>>> print(a)

True

Не (not)

Этот оператор инвертирует [**булевые значения**](https://pythonru.com/osnovy/tipy-dannyh-v-python) выражения. True превращается в False и наоборот. В примере внизу булево значение 0 — False. Поэтому оно превращается в True.  
**Пример:**

>>> a = not(0)

>>> print(a)

True

## Операторы принадлежности

Эти операторы проверяют, является ли значение частью последовательности. Последовательность может быть списком, строкой или [**кортежем**](https://pythonru.com/uroki/kortezhi-tuple-uroki-po-python-dlja-nachinajushhih). Есть всего два таких оператора: in и not in.

В (in)

Проверяет, является ли значение членом последовательности. В этом примере видно, что строки fox нет в списке питомцев. Но cat — есть, поэтому она возвращает True. Также строка me является подстрокой disappointment. Поэтому она вернет True.  
**Пример:**

>>> pets=['dog','cat', 'ferret']

>>> 'fox' in pets

False

>>> 'cat' in pets

True

>>> 'me' in 'disappointment'

True

Нет в (not in)

Этот оператор проверяет, НЕ является ли значение членом последовательности.  
**Пример:**

>>> 'pot' not in 'disappointment'

True

# Pattern matching. Конструкция match

Начиная с версии 3.10 в языке Python появилась такая функциональность как pattern matching (сопоставление шаблонов). Pattern matching представляет применение конструкции match, которая позволяет сопоставить выражение с некоторым шаблоном. И если выражение соответствует шаблону, то выполняются определенные действия. В этом смысле конструкция match похожа на конструкцию if/else/elif, которая выполняет определенные действия в зависимости от некоторого условия. Однако функциональность match гораздо шире - она также позволяет извлечь данные из составных типов и применить действия к различным частям объектов.

Конструкция match имеет следующее формальное определение:

match выражение:

case шаблон\_1:

действие\_1

case шаблон\_2:

действие\_2

................

case шаблон\_N:

действие\_N

case \_:

действие\_по\_умолчанию

Пример 1

language = **"english"**language = **"russian"  
  
match** language:  
 **case "russian"**:  
 print(**"Привет"**)  
 **case "english"**:  
 print(**"Hello"**)  
 **case "german"**:  
 print(**"Hallo"**)

Причем если выражение из match не соответствует ни одному из шаблонов case, то соответственно ни один из этих блоков case не выполняется.

Если необходимо, чтобы при несовпадении значений (если ни один из шаблонов case не соответствует выражению match) выполнялись некоторые действия по умолчанию, то в этом случае применяется шаблон \_ (прочерк):

**case** \_:  
 print(**"Undefined"**)

Но также можно определить блок case, который позволяет сравнивать сразу с несколькими знечениями. В этом случае значения разделяются вертикальной чертой:

**case "american english"** | **"british english"** | **"english"**:  
 print(**"Hello"**)

Пример 2

|  |  |
| --- | --- |
| http\_status = 400  if http\_status == 400:  print("Bad Request")  elif http\_status == 403:  print("Forbidden")  elif http\_status == 404:  print("Not Found")  else:  print("Other") | http\_status = 400  match http\_status:  case 400:  print("Bad Request")  case 403:  print("Forbidden")  case 404:  print("Not Found")  case \_:  print("Other") |
| day = "Monday"  match day:  case "Sunday" : print("Take it easy")  case "Monday" : print("Go to work")  case "Tuesday" : print("Work + Hobbies")  case "Wednesday" : print("Meetings")  case "Thursday" : print("Presentations")  case "Friday" : print("Interviews and party")  case "Saturday" : print("Time to do sports") | day = "Monday"  match day:  case "Saturday" | "Sunday":  print("Weekend")  case "Monday" | "Tuesday" | "Wednesday" | "Thursday" | "Friday":  print("Work") |
| cmd = 10    match cmd:  case str() as command:  print(f"строковая команда: {command}")  case int() as command:  print(f"целочисленная команда: {command}")  case bool() as command:  print(f"булева команда: {command}")  case \_: # wildcard  print(f"другая команда") | сmd = "c\_top"    match cmd:  case str() as command if len(command) < 10 and command[0] == 'c':  print(f"строковая команда: {command}")  case bool() as command:  print(f"булева команда: {command}")  case int() as command if 0 <= command <= 9:  print(f"целочисленная команда: {command}")  case \_: # wildcard  print(f"другая команда") |
| case int() | float() as command if 0 <= command <= 9: |  |

В качестве шаблонов в pathern matching в Python могут выступать кортежи, массивы, словари и даже классы.