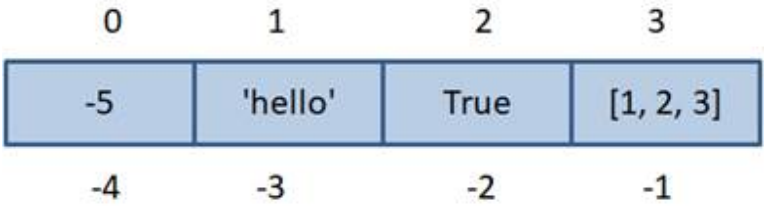
Кортежи (tuple) и их методы

На этом занятии мы с вами познакомимся с еще одной новой коллекцией данных – кортежами. Что такое кортеж? Это упорядоченная, но неизменяемая коллекция произвольных данных. Они, в целом, аналогичны спискам, но в отличие от них, элементы кортежей менять нельзя – это неизменяемый тип данных.



Для задания кортежа достаточно перечислить значения через запятую:

```
a = 1,2
```

или, что то же самое, в круглых скобках:

```
a = (1, 2, 3)
```

Но, обратите внимание, при определении кортежа с одним значение, следует использовать такой синтаксис:

b = 1, или b = (1,)

Здесь висячая запятая указывает, что единицу следует воспринимать как первый элемент кортежа, а не как число 1. Без запятой – это будет уже просто число один. Вот это нужно очень хорошо запомнить, начинающие программисты здесь часто делают ошибку, не прописывая висячую запятую.

Далее, мы можем переменным сразу присвоить соответствующие значения из кортежа, перечисляя их через запятую:

```
x, y = (1, 2)
```

но, если число переменных не будет совпадать с числом элементов кортежа:

то возникнет ошибка.

```
x, y = (1, 2, 3)
```

Длину кортежа (число его элементов) можно определить с помощью известной уже вам функции:

```
len(a)
```

А доступ к его элементам осуществляется также как и к элементам списков: **по индексу и по срезу.**

О срезах мы с вами уже подробно говорили, когда рассматривали списки, здесь все абсолютно также, кроме одной операции – взятия полного среза:

```
b = a[:]
```

В случае с кортежем здесь не создается его копии – это будет ссылка на тот же самый объект:

Напомню, что для списков эта операция приводит к их копированию. Этот момент нужно также очень хорошо запомнить: для кортежей –возвращается тот же самый объект, а для списков – создается копия.

Некоторые из вас сейчас могут задаваться вопросом: зачем было придумывать кортежи, если списки способны выполнять тот же самый функционал? И даже больше – у них можно менять значения элементов, в отличие от кортежей? Да, все верно, по функциональности списки шире кортежей, но у последних все же есть свои преимущества.

Во-первых, то, что кортеж относится к неизменяемому типу данных, то мы можем использовать его, когда необходимо «запретить» программисту менять значения элементов списка. Например, вот такая операция: `a[1] = 100`

приведет к ошибке. Менять значения кортежей нельзя. Во-вторых, кортежи можно использовать в качестве ключей у словарей, например, так:

```
d = {}
d[a] = "кортеж"
```

Напомню, что списки как ключи применять недопустимо, так как список – это изменяемый тип, а ключами могут быть только неизменяемые типы и кортежи здесь имеют преимущество перед списками.

В-третьих, кортеж занимает меньше памяти, чем такой же список.

Чтобы создать пустой кортеж можно просто записать круглые скобки: `a = ()`

или воспользоваться специальной встроенной функцией: `b = tuple()`

Но здесь сразу может возникнуть вопрос: зачем создавать пустой кортеж, если он относится к неизменяемым типам данных? Слово **неизменяемый** наводит на мысль, что вид кортежа остается неизменным. Но это не совсем так. В действительности, мы лишь не можем менять уже существующие элементы в кортеже, но можем создавать новые, используя оператор +, например:

```
a = ()
a = a +  (1,)
```

или для добавления данных в начало кортежа: `a = (2, 3) + a`

Также можно добавить вложенный кортеж: `a += (("a", "hello"),)`

или дублировать элементы кортежа через оператор *, подобно спискам:

```
b = (0,)*10
b = ("hello", "world") * 5
```

Как видите, все эти операции вполне допустимы. А вот удалять отдельные его элементы уже нельзя. Если мы попытаемся это сделать: `del a[1]`

то возникнет ошибка. Правда, можно удалить объект целиком: `del a`

тогда кортеж просто перестанет существовать, не будет даже пустого кортежа, здесь объект удаляется целиком.

Далее, с помощью функции tuple() можно создавать кортежи на основе любого итерируемого объекта, например, списков и строк:

```
a = tuple([1,2,3])
a = tuple("Привет мир")
```

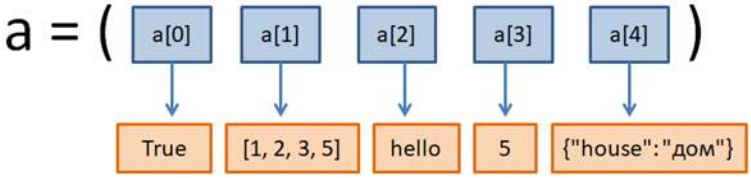
А если нам нужно превратить кортеж в список, то подойдет известная нам функция list(), которая формирует список также из любого итерируемого объекта:

```
t = (1,2,3)
lst = list(t)
```

Так как кортеж – это перебираемый, то есть, итерируемый объект, то никто не мешает нам превращать его в список.

Как я отмечал в самом начале, кортежи могут хранить самые разные данные:

```
a = (True, [1,2,3], "hello", 5, {"house": "дом"})
```



Причем, смотрите, если обратиться, например, к списку: `a[1]`

то это изменяемый объект, следовательно, его значение даже в кортеже мы можем спокойно менять: `a[1].append("5")`

То есть, неизменность кортежа относится к его структуре элементов и значениям ссылок на объекты. Но, если сами объекты, при этом, могут меняться, то мы можем легко это делать. То есть, обращаясь ко второму элементу кортежа, мы, фактически, имеем список, с которым возможны все его операции без каких-либо ограничений. Я, думаю, это должно быть понятно?

В заключение этого занятия рассмотрим два метода, которые часто используются в кортежах – это:

tuple.count(значение) – возвращает число найденных элементов с указанным значением;

tuple.index(значение[, start[, stop]]) – возвращает индекс первого найденного элемента с указанным значением (start и stop –необязательные параметры, индексы начала и конца поиска).

Эти методы работают так же, как и у списков. Пусть у нас есть кортеж:

```
a = ("abc", 2, [1,2], True, 2, 5)
```

И мы хотим определить число элементов со значениями:

```
a.count("abc")
a.count(2)
```

Как видите, метод count() возвращает число таких элементов в кортеже. Если же элемент не будет найден:

```
a.count("a")
```

то возвращается 0. Следующий метод:

```
a.index(2)
```

возвращает индекс первого найденного элемента с указанным значением. Если значение не найдено:

```
a.index(3)
```

то этот метод приводит к ошибке. Поэтому, прежде чем его использовать, лучше проверить, а есть ли такое значение вообще в кортеже:

```
3 in a
```

Второй необязательный параметр

```
a.index(2, 3)
```

позволяет задавать индекс начала поиска. В данном случае поиск будет начинаться с третьего индекса. А последний третий аргумент:

```
a.index(2, 3, 5)
```

определяет индекс, до которого идет поиск (не включая его). Если же записать вот так:

```
a.index(2, 3, 4)
```

то возникнет ошибка, т.к. в поиск будет включен только 3-й индекс и там нет значения 2.