

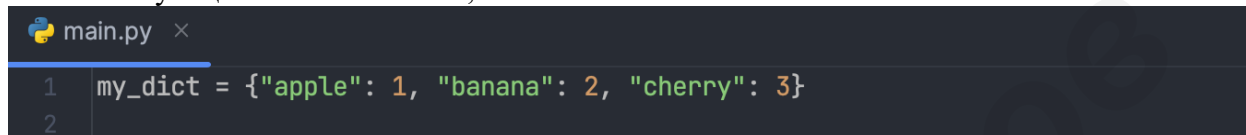
## ТЕМА 6. БАЗОВЫЕ КОЛЛЕКЦИИ: СЛОВАРИ, КОРТЕЖИ.

### 6.1. Словари

Словарь (Dictionary): Словарь представляет неупорядоченную коллекцию пар ключ-значение. Каждый элемент в словаре имеет уникальный ключ, и к нему можно обратиться по ключу для получения значения. Словари в Python обозначаются фигурными скобками {} и используются для хранения и доступа к данным с помощью ключей.

Пример создания словаря в Python представлен на рисунке 6.1.

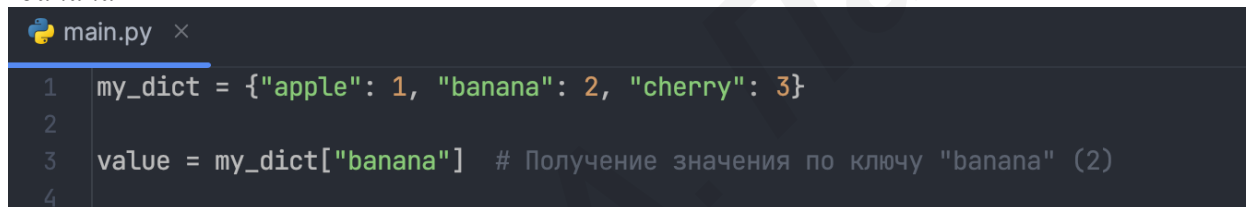
В этом примере `my_dict` - это словарь с ключами "apple", "banana" и "cherry", и соответствующими значениями 1, 2 и 3.



```
main.py x
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2
```

Рис. 6.1. Пример создания словаря в Python

Доступ к значениям словаря осуществляется по ключу, представлен на рисунке 6.2. В этом примере `my_dict["banana"]` возвращает значение, связанное с ключом "banana".



```
main.py x
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2
3 value = my_dict["banana"] # Получение значения по ключу "banana" (2)
4
```

Рис. 6.2. Доступ к значениям словаря

### 6.2. Виды создания словарей

В Python существует несколько способов создания словарей. Рассмотрим некоторые из них:

- Литералы словаря

С использованием фигурных скобок {} и указанием пар ключ-значение через двоеточие : (Рисунок 6.3.)

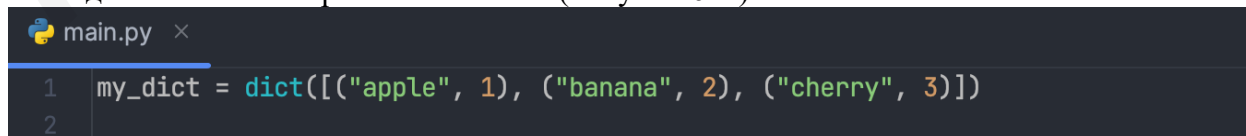


```
main.py x
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2
```

Рис. 6.3. Литералы словаря

- Функция `dict()`

С использованием функции `dict()` и передачей итерируемого объекта, содержащего последовательность пар ключ-значение (Рисунок 6.4.)

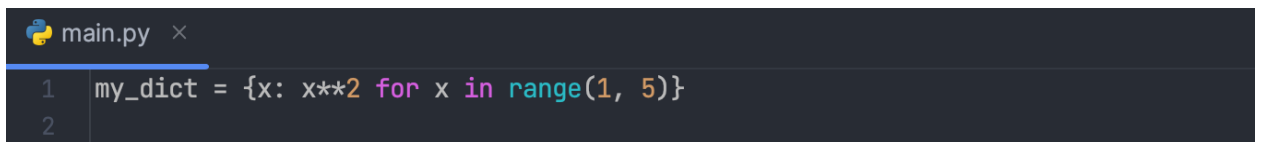


```
main.py x
1 my_dict = dict([("apple", 1), ("banana", 2), ("cherry", 3)])
2
```

Рис. 6.4. Функция `dict()`

- Компреихеншн словаря (*Dictionary comprehension*)

Позволяет создавать словари на основе выражений и итерирования (Рисунок 6.5.)



```

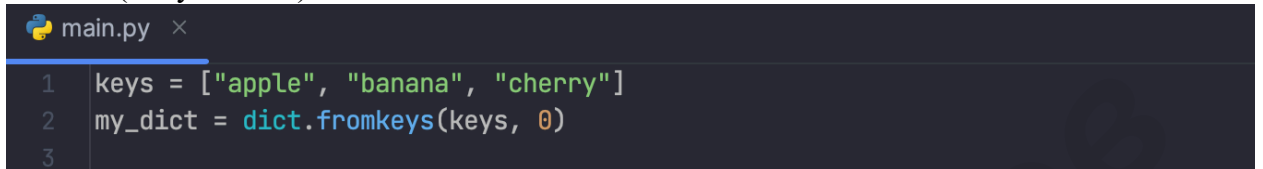
1 my_dict = {x: x**2 for x in range(1, 5)}
2

```

Рис. 6.5. Комprehеншн словаря

- Метод *fromkeys()*:

Создает новый словарь с указанными ключами и опциональным значением для всех ключей (Рисунок 6.6.)



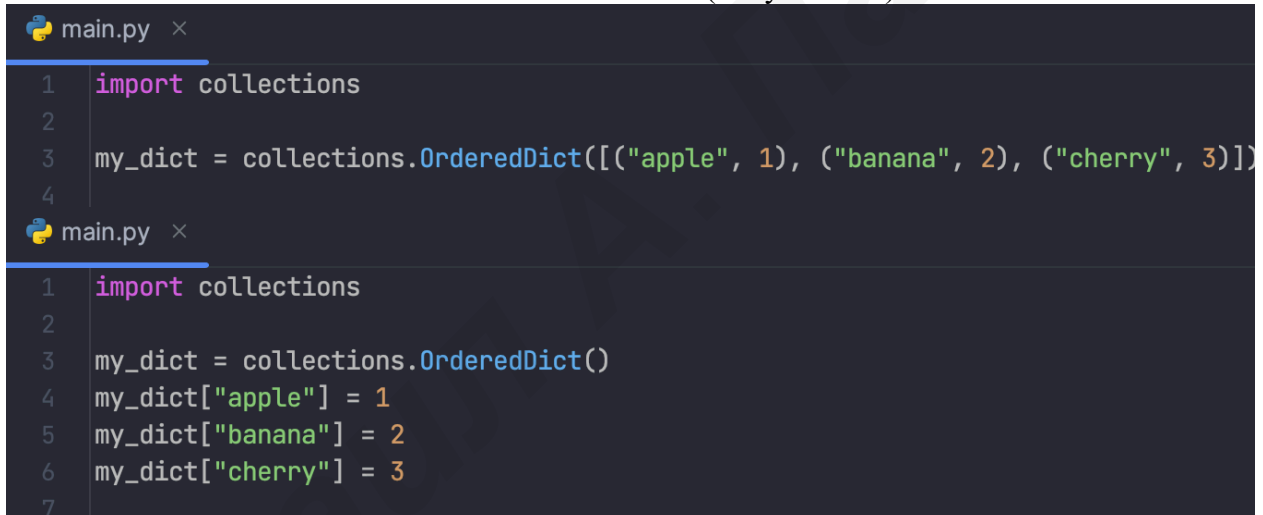
```

1 keys = ["apple", "banana", "cherry"]
2 my_dict = dict.fromkeys(keys, 0)
3

```

Рис. 6.6. Метод *fromkeys()*:

Обратите внимание, что при создании словаря с помощью литералов или функции *dict()*, порядок элементов может не сохраняться, так как словари в Python являются неупорядоченной коллекцией. Если вам требуется сохранить порядок элементов, вы можете использовать класс *collections.OrderedDict* (Рисунок 6.7.)



```

1 import collections
2
3 my_dict = collections.OrderedDict([("apple", 1), ("banana", 2), ("cherry", 3)])
4

```

```

1 import collections
2
3 my_dict = collections.OrderedDict()
4 my_dict["apple"] = 1
5 my_dict["banana"] = 2
6 my_dict["cherry"] = 3
7

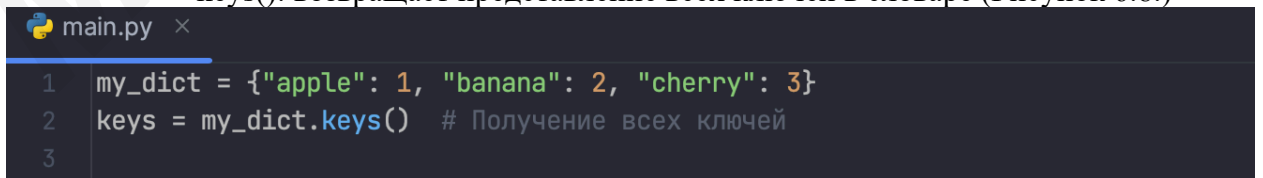
```

Рис. 6.7. Класс *collections.OrderedDict*

### 6.3. Методы словарей

В Python словари (dict) предоставляют ряд встроенных методов для выполнения различных операций. Вот некоторые из наиболее часто используемых методов словарей:

- *keys()*: возвращает представление всех ключей в словаре (Рисунок 6.8.)



```

1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 keys = my_dict.keys() # Получение всех ключей
3

```

Рис. 6.8. Метод *keys()*

- *values()*: возвращает представление всех значений в словаре (Рисунок 6.9.)

```

main.py x
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 values = my_dict.values() # Получение всех значений
3

```

Рис. 6.9. Метод *values()*

- *items()*: возвращает представление всех пар ключ-значение в словаре (Рисунок 6.10.)

```

main.py x
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 items = my_dict.items() # Получение всех пар ключ-значение
3

```

Рис. 6.10. Метод *items()*

- *get(key, default)*: возвращает значение, связанное с указанным ключом. Если ключ не найден, возвращается значение по умолчанию (Рисунок 6.11.)

```

main.py x
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 value = my_dict.get("apple", 0) # Получение значения по ключу "apple" (1)
3

```

Рис. 6.11. Метод *get(key, default)*

- *pop(key, default)*: удаляет элемент с указанным ключом и возвращает его значение. Если ключ не найден, возвращается значение по умолчанию (Рисунок 6.12.)

```

main.py x
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 value = my_dict.pop("banana", 0) # Удаление элемента с ключом "banana" и воз
3

```

Рис. 6.12. Метод *pop(key, default)*

- *popitem()*: удаляет и возвращает произвольную пару ключ-значение из словаря (Рисунок 6.13.)

```

main.py x
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 key, value = my_dict.popitem() # Удаление и возвращение произвольной пары кл
3

```

Рис. 6.13. Метод *popitem()*

- *update(other\_dict)*: обновляет словарь, добавляя пары ключ-значение из другого словаря (Рисунок 6.14.)

```

main.py x
1 my_dict = {"apple": 1, "banana": 2}
2 other_dict = {"cherry": 3, "date": 4}
3 my_dict.update(other_dict) # Обновление словаря

```

Рис. 6.14. Метод *update(other\_dict)*

- `clear()`: удаляет все элементы из словаря (Рисунок 6.15.)

```
main.py ×
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 my_dict.clear() # Очистка словаря
3
```

Рис. 6.15. Метод `clear()`

- `copy()`: создает и возвращает копию словаря (Рисунок 6.16.)

```
main.py ×
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 copy_dict = my_dict.copy() # Создание копии словаря
3
```

Рис. 6.16. Метод `copy()`

Это лишь некоторые из методов словарей в Python. С помощью них можно выполнять различные операции, включая получение ключей, значений и пар ключ-значение, добавление и удаление элементов, обновление словарей и многое другое.

#### 6.4. Обращения к элементам словаря.

В Python доступ к элементам словаря осуществляется по ключу. Поскольку словари являются неупорядоченными коллекциями, они не поддерживают индексацию элементов, как это делается в случае со списками. Вот несколько способов обращения к элементам словаря:

- **Использование квадратных скобок []:**

Чтение значения по ключу (Рисунок 6.17.)

```
main.py ×
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 value = my_dict["apple"] # Получение значения по ключу "apple" (1)
3
```

Рис. 6.17. Чтение значения по ключу

Обновление значения по ключу (Рисунок 6.18.)

```
main.py ×
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 my_dict["banana"] = 5 # Обновление значения по ключу "banana"
3
```

Рис. 6.18. Обновление значения по ключу

- **Метод `get()`:**

Чтение значения по ключу с использованием метода `get()`. Если ключ не найден, можно указать значение по умолчанию (Рисунок 6.19.)

```
main.py ×
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 value = my_dict.get("apple") # Получение значения по ключу "apple" (1)
3 value = my_dict.get("date", 0) # Получение значения по ключу "date" с значен
4
```

Рис. 6.19. Метод `get()`

- Метод `keys()`, `values()` и `items()`:

Итерация по ключам, значениям и парам ключ-значение (Рисунок 6.20.)

```
main.py ×
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2 for key in my_dict.keys():
3     print(key) # Вывод ключей
4
5 for value in my_dict.values():
6     print(value) # Вывод значений
7
8 for key, value in my_dict.items():
9     print(key, value) # Вывод пар ключ-значение
10

/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/
apple
banana
cherry
1
2
3
apple 1
banana 2
cherry 3

Process finished with exit code 0
```

Рис. 6.20. Метод `keys()`, `values()` и `items()`

Обратите внимание, что при обращении к несуществующему ключу будет возникать исключение `KeyError`. Чтобы избежать этой ошибки, можно использовать метод `get()` с опциональным значением по умолчанию или проверять наличие ключа с помощью оператора `in` перед обращением к нему (Рисунок 6.21.)

```
main.py ×
1 my_dict = {"apple": 1, "banana": 2, "cherry": 3}
2
3 if "apple" in my_dict:
4     value = my_dict["apple"]
5 else:
6     value = 0
7
8 # Или
9
10 # value = my_dict.get("apple", 0)
11
```

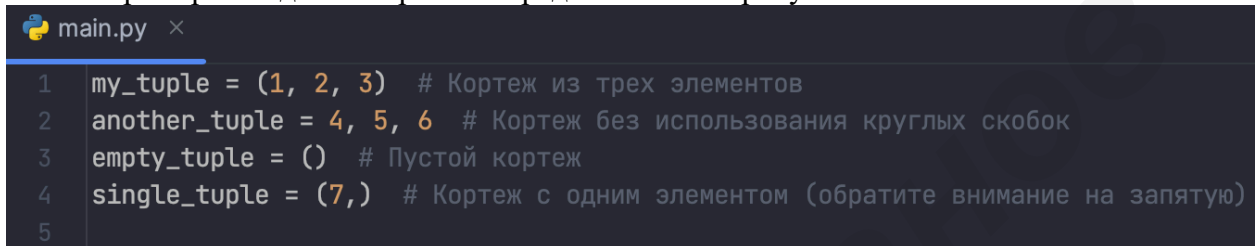
Рис. 6.21. Пример обращения к несуществующему ключу

## 6.5. Кортежи

В Python кортеж (tuple) представляет собой неизменяемую упорядоченную коллекцию объектов. Кортежи очень похожи на списки, но имеют несколько отличий. Основные особенности кортежей:

- **Неизменяемость:** кортежи не могут быть изменены после создания. Это означает, что нельзя добавлять, удалять или изменять элементы кортежа. Однако, если элементы кортежа являются изменяемыми объектами, то их состояние может быть изменено.
- **Упорядоченность:** элементы в кортеже упорядочены и доступ к ним осуществляется по индексу.
- **Доступ к элементам:** к элементам кортежа можно обращаться по индексу, так же как и к элементам списка.
- **Использование круглых скобок:** для создания кортежа используются круглые скобки `()` или иногда их можно опустить при объявлении кортежа.

Примеры создания кортежей представлены на рисунке 6.22.



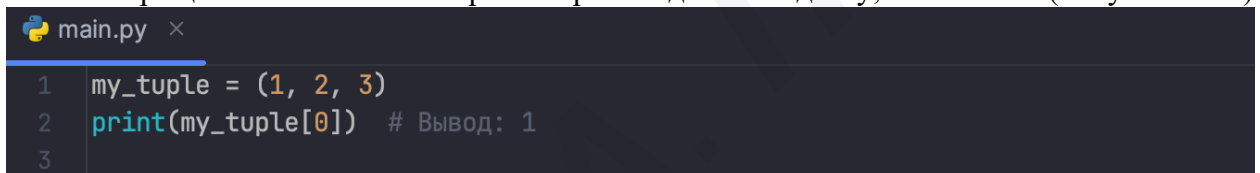
```

main.py x
1 my_tuple = (1, 2, 3) # Кортеж из трех элементов
2 another_tuple = 4, 5, 6 # Кортеж без использования круглых скобок
3 empty_tuple = () # Пустой кортеж
4 single_tuple = (7,) # Кортеж с одним элементом (обратите внимание на запятую)
5

```

Рис. 6.22. Примеры создания кортежей

Обращение к элементам кортежа происходит по индексу, начиная с 0 (Рисунок 6.23.)



```

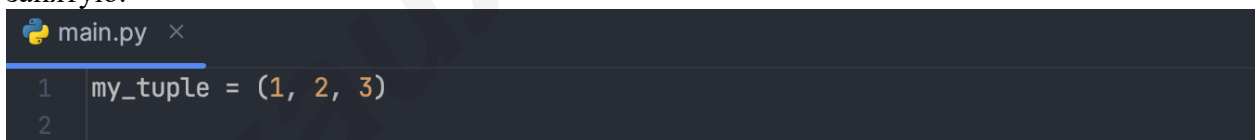
main.py x
1 my_tuple = (1, 2, 3)
2 print(my_tuple[0]) # Вывод: 1
3

```

Рис. 6.23. Обращение к элементам кортежа происходит по индексу

- **Использование круглых скобок** (Рисунок 6.24.)

С помощью круглых скобок `()` можно объявить кортеж, указав его элементы через запятую.



```

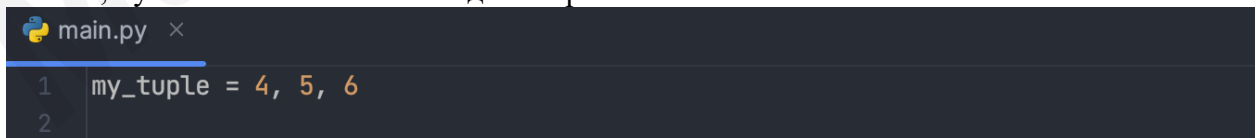
main.py x
1 my_tuple = (1, 2, 3)
2

```

Рис. 6.24. Использование круглых скобок

- **Без использования круглых скобок** (Рисунок 6.25.)

Если элементы кортежа перечислены через запятую без использования круглых скобок, Python автоматически создаст кортеж.



```

main.py x
1 my_tuple = 4, 5, 6
2

```

Рис. 6.25. Без использования круглых скобок

- **Использование функции `tuple()`** (Рисунок 6.26.)

Функция `tuple()` может быть использована для создания кортежа из итерируемого объекта, например, списка.



```

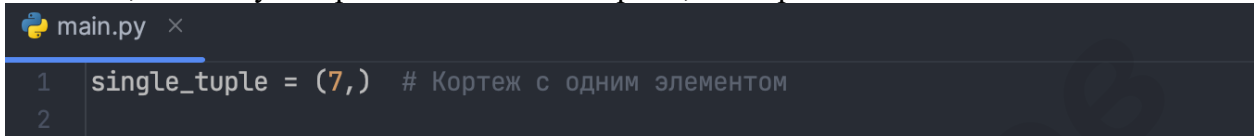
main.py x
1 my_list = [1, 2, 3]
2 my_tuple = tuple(my_list)
3

```

Рис. 6.26. Использование функции *tuple()*

- Кортеж с одним элементом (Рисунок 6.27.)

Если кортеж содержит только один элемент, необходимо добавить запятую после элемента, чтобы Python распознал его как кортеж, а не простое значение.



```

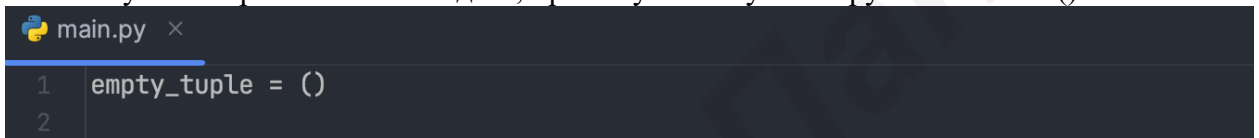
main.py x
1 single_tuple = (7,) # Кортеж с одним элементом
2

```

Рис. 6.27. Кортеж с одним элементом

- Пустой кортеж (Рисунок 6.28.)

Пустой кортеж можно создать, просто указав пустые круглые скобки ().



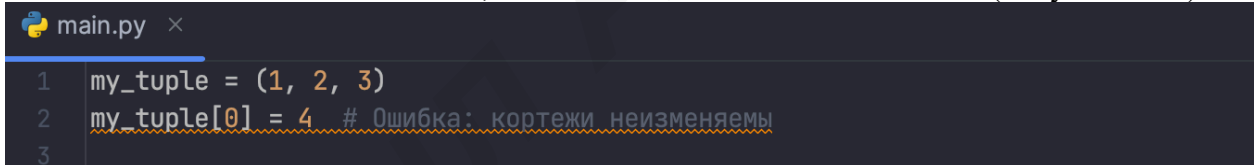
```

main.py x
1 empty_tuple = ()
2

```

Рис. 6.28. Пустой кортеж

Обратите внимание, что кортежи являются неизменяемыми, поэтому после создания нельзя добавлять, удалять или изменять элементы кортежа. Однако, если элементы кортежа являются изменяемыми объектами, их состояние может быть изменено (Рисунок 6.29.)



```

main.py x
1 my_tuple = (1, 2, 3)
2 my_tuple[0] = 4 # Ошибка: кортежи неизменяемы
3

```

Рис. 6.29. Ошибка: кортежи неизменяемы

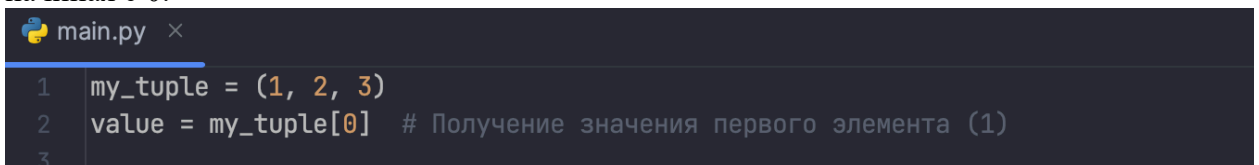
Кортежи широко используются для представления упорядоченных наборов данных, которые не должны быть изменяемыми. Они также могут быть использованы для передачи нескольких значений в качестве возвращаемого значения функции или в качестве аргументов функции, когда требуется сохранить порядок передачи данных.

## 6.6. Операции над кортежами

Кортежи в Python являются неизменяемыми, что означает, что после создания нельзя изменять их элементы. Однако, существуют некоторые операции, которые можно выполнять над кортежами:

- Доступ к элементам (Рисунок 6.30.)

Как и в случае со списками, доступ к элементам кортежа осуществляется по индексу, начиная с 0.



```

main.py x
1 my_tuple = (1, 2, 3)
2 value = my_tuple[0] # Получение значения первого элемента (1)
3

```

Рис. 6.30. Получение значения первого элемента

- Срезы (Рисунок 6.31.)

Кортежи поддерживают операцию среза, позволяющую получить подкортеж из определенного диапазона элементов.

```
main.py x
1 my_tuple = (1, 2, 3, 4, 5)
2 sub_tuple = my_tuple[1:4] # Получение подкортежа с индексами 1, 2, 3 (2, 3, 4)
3
```

Рис. 6.31. Получение подкортежа с индексами

- Конкатенация (Рисунок 6.32.)

Два кортежа можно объединить с помощью оператора +, создавая новый кортеж, содержащий элементы обоих кортежей.

```
main.py x
1 tuple1 = (1, 2, 3)
2 tuple2 = (4, 5, 6)
3 merged_tuple = tuple1 + tuple2 # Объединение кортежей (1, 2, 3, 4, 5, 6)
4
```

Рис. 6.32. Объединение кортежей

- Длина кортежа (Рисунок 6.33.)

Функция len() возвращает количество элементов в кортеже.

```
main.py x
1 my_tuple = (1, 2, 3)
2 length = len(my_tuple) # Получение длины кортежа (3)
3
```

Рис. 6.33. Получение длины кортежа

- Повторение (Рисунок 6.34.)

Кортеж можно повторить определенное количество раз с помощью оператора умножения \*.

```
main.py x
1 my_tuple = (1, 2)
2 repeated_tuple = my_tuple * 3 # Повторение кортежа 3 раза (1, 2, 1, 2, 1, 2)
3
```

Рис. 6.34. Повторение кортежа 3 раза

- Проверка наличия элемента (Рисунок 6.35.)

Оператор in позволяет проверить, содержится ли элемент в кортеже.

```
main.py x
1 my_tuple = (1, 2, 3)
2 contains = 2 in my_tuple # Проверка наличия элемента 2 в кортеже (True)
3
```

Рис. 6.35. Проверка наличия элемента 2 в кортеже (True)

Обратите внимание, что все эти операции не изменяют исходный кортеж, а создают новые кортежи или возвращают значения.

## 6.7. Кортеж как параметр и результат функций



В Python кортежи могут использоваться как параметры и результаты функций. Это позволяет передавать и возвращать несколько значений вместе.


- Кортеж как параметр функции. Вы можете передать кортеж в качестве параметра функции, чтобы передать несколько значений одновременно (Рисунок 6.36.)



```
main.py x
1 usage
2 def my_function(my_tuple):
3     for value in my_tuple:
4         print(value)
5
6 my_tuple = (1, 2, 3)
7 my_function(my_tuple)
8
/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/la
1
2
3
Process finished with exit code 0
```

Рис. 6.36. Кортеж как параметр функции

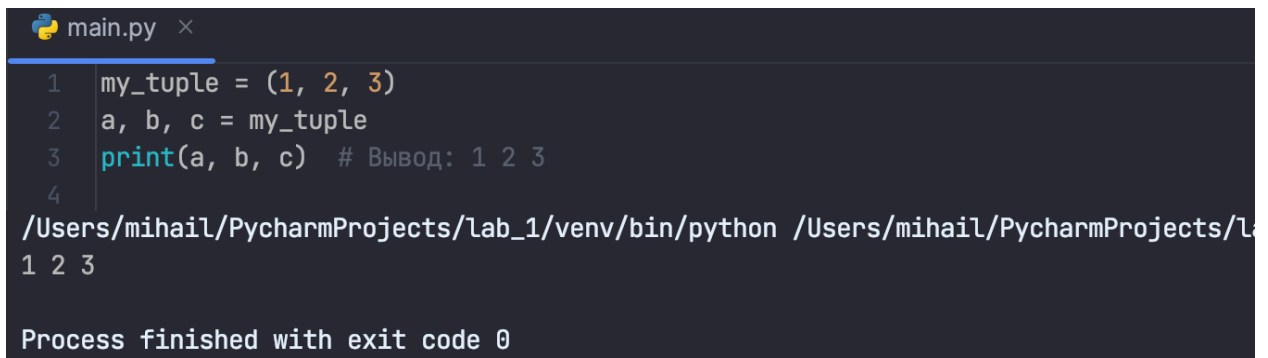
- Кортеж в качестве возвращаемого значения функции. Функция может возвращать кортеж вместо одного значения. Это позволяет вернуть несколько результатов из функции (Рисунок 6.37.)



```
main.py x
1 usage
2 def my_function():
3     return 1, 2, 3
4
5 result_tuple = my_function()
6 print(result_tuple) # Вывод: (1, 2, 3)
7
/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/la
(1, 2, 3)
Process finished with exit code 0
```

Рис. 6.37. Кортеж в качестве возвращаемого значения функции

- Распаковка кортежа. Кортеж можно распаковать для присваивания его элементов отдельным переменным (Рисунок 6.38.)

A screenshot of a Python IDE window titled 'main.py'. The code consists of four lines: 1. 'my\_tuple = (1, 2, 3)', 2. 'a, b, c = my\_tuple', 3. 'print(a, b, c) # Вывод: 1 2 3', and 4. An empty line. Below the code, the terminal output shows the file path '/Users/mihail/PycharmProjects/lab\_1/venv/bin/python /Users/mihail/PycharmProjects/lab\_1/main.py' followed by the output '1 2 3'. At the bottom, it says 'Process finished with exit code 0'.

```
main.py ×
1 my_tuple = (1, 2, 3)
2 a, b, c = my_tuple
3 print(a, b, c) # Вывод: 1 2 3
4
/Users/mihail/PycharmProjects/lab_1/venv/bin/python /Users/mihail/PycharmProjects/lab_1/main.py
1 2 3
Process finished with exit code 0
```

Рис. 6.38. Распаковка кортежа

Кортежи как параметры функций и их использование в качестве возвращаемых значений являются полезным инструментом для передачи и получения нескольких значений одновременно. Они позволяют группировать связанные данные в структуры и работать с ними как с единым объектом.

Кортежи в Python имеют многочисленные применения и являются полезными в различных ситуациях. Они могут быть использованы для представления неизменяемых данных, передачи нескольких значений в функции, возвращения нескольких результатов из функции, структурирования данных и обеспечения целостности данных в программе.