**Базовые структуры данных - Негизги маалымат структуралары**

Бул маалымат структуралары Python тилинин көпчүлүк операциялары жана алгоритмдери үчүн негиз болуп саналат жана тилдин иштешин түшүнүү үчүн ачкыч болуп саналат.

**Списки (list)**

1. Тизмелер-Бул ар кандай типтеги элементтерди камтышы мүмкүн болгон объекттердин иреттелген жыйнагы.

* empty\_list = []
* fruits = ["apple", "banana", "cherry", "date"]
* mixed\_list = [1, "apple", 3.14, True]

2. Тизме элементтери 0дөн баштап индекстелет. Элементтерге индекстери боюнча кире аласыз.

* first\_fruit = fruits[0] # apple
* third\_fruit = fruits[2] # cherry

3. Терс индекстер тизменин аягындагы нерселерге кайрылууга мүмкүндүк берет.

* last\_fruit = fruits[-1] # date

Списки в Python — это изменяемые объекты.

4. Элементтерди кошуу:

* fruits.append("elderberry") # добавляет элемент в конец списка
* fruits.insert(1, "apricot") # вставляет "apricot" на вторую позицию

5. Элементтерди алып салуу:

* fruits.remove("banana") # удаляет первый элемент "banana"
* popped = fruits.pop(2) # удаляет и возвращает элемент по индексу (в данном случае "cherry")

6. Элементти өзгөртүү:

* fruits[1] = "blueberry"

7. Позволяют получать подсписки из списка.

* sublist = fruits[1:3] # ['banana', 'cherry']

**list.append(x):** добавляет элемент в конец списка.

**list.extend(iterable):** расширяет список, добавляя в конец все элементы из передаваемого списка.

**list.insert(i, x):** вставляет элемент на указанную позицию.

**list.remove(x):** удаляет первый элемент списка, имеющий значение x.

**list.pop([i]):** удаляет элемент на указанной позиции и возвращает его. Если индекс не указан, удаляет последний элемент.

**list.index(x[, start[, end]]):** возвращает индекс первого элемента со значением x.

**list.count(x):** возвращает количество элементов со значением x.

**list.sort(key=None, reverse=False):** сортирует элементы списка.

**list.reverse():** инвертирует порядок элементов в списке.

**Применение списков в бэкенд-разработке**

1. Динамическое хранение данных: Поскольку списки могут изменяться, они идеально подходят для ситуаций, когда требуется динамически добавлять или удалять данные, например, списки пользователей онлайн или товары в корзине.
2. Буферизация: Списки могут использоваться для хранения временных данных, например, последних N записей журнала для быстрого доступа.
3. Обработка данных: При анализе или обработке данных, например, при извлечении данных из базы данных или при работе с API, списки могут использоваться для итерации, сортировки, фильтрации и трансформации данных.
4. Стеки и очереди: Списки можно использовать для создания структур данных, таких как стеки и очереди, что полезно для определенных алгоритмов или для управления рабочими процессами.

**Кортежи (tuple)**

1. Кортеждер тизмелерге окшош объекттердин иреттелген жыйнагы. Негизги айырмачылык-кортеждер өзгөрүлбөйт, демек, кортежди жараткандан кийин элементтерди өзгөртүүгө, кошууга же жок кылууга болбойт.

* empty\_tuple = ()
* single\_element\_tuple = ("apple",) # обратите внимание на запятую
* fruits = ("apple", "banana", "cherry", "date")

2. Ошондой эле, тизмелердегидей эле, кортеждин элементтери 0дөн баштап индекстелет.

* first\_fruit = fruits[0] # apple
* third\_fruit = fruits[2] # cherry

3. Кортеждин элементтерин өзгөртө албай турганыңызды эсиңизден чыгарбаңыз. Эгер аракет кылсаңыз, ката кетириңиз.

* # fruits[1] = "blueberry" # вызовет ошибку

4. Однако вы можете переопределить весь кортеж:

* fruits = ("apple", "blueberry", "cherry", "date")

5. Кортеждер өзгөрүлбөгөндүктөн, тизмелерге салыштырмалуу алардын ыкмалары азыраак. Бирок алар менен дагы эле кээ бир операцияларды жасоого болот:

Конкатенация:

* numbers = (1, 2, 3)
* letters = ("a", "b", "c")
* combo = numbers + letters # (1, 2, 3, "a", "b", "c")

Умножение:

* repeated = numbers \* 3 # (1, 2, 3, 1, 2, 3, 1, 2, 3)

Элементтин бар экендигин текшерүү:

* if "apple" in fruits:
  + print("Apple is in the tuple!")

6. Зачем использовать кортежи?

1. Безопасность: Так как кортежи неизменяемы, они могут служить "защитой от записи", предотвращая случайные изменения данных.
2. Хешируемость: Поскольку кортежи являются неизменяемыми, они могут использоваться в качестве ключей для словарей, тогда как списки — нет.
3. Оптимизация: Кортежи могут быть более эффективными с точки зрения памяти и производительности по сравнению со списками для небольших коллекций.
4. Семантика: Использование кортежа может сигнализировать, что набор данных не предназначен для изменения (например, координаты, RGB-цвета и т. д.).
5. Представление фиксированных наборов данных: Как упоминалось выше, кортежи идеально подходят для представления данных, которые не должны изменяться, например, дни недели, месяцы, координаты и т. д.

**Множества (set)**

1. Топтомдор уникалдуу элементтердин иреттелбеген жыйнагы. Демек, ар бир элемент топтомдо бир гана жолу пайда болушу мүмкүн.

* empty\_set = set() # создание пустого множества
* fruits = {"apple", "banana", "cherry", "apple"} # "apple" будет добавлен только один раз

2. Элемент кошуу:

* fruits.add("date")

3. Элементти алып салуу:

* fruits.remove("banana") # вызывает ошибку, если элемент не найден
* fruits.discard("banana") # не вызывает ошибку, если элемент не найден

4. Элементтин бар экендигин текшерүү:

* if "cherry" in fruits:
  + print("Cherry is in the set!")

5. Топтомдун узундугу:

* length = len(fruits)

6. Биригүү:

* fruits1 = {"apple", "banana", "cherry"}
* fruits2 = {"apple", "date", "elderberry"}
* union\_set = fruits1.union(fruits2)

7. Кесилиш:

* intersection\_set = fruits1.intersection(fruits2) # вернет {"apple"}

8. Айырма:

* difference\_set = fruits1.difference(fruits2) # вернет {"banana", "cherry"}

9. Применение множеств в бэкенд-разработке

1. Удаление дубликатов: Множества идеально подходят для быстрого удаления дубликатов из списка.

* unique\_fruits = set(fruits\_list)

1. Проверка принадлежности: Проверка наличия элемента в множестве происходит гораздо быстрее, чем в списке.
2. Операции над множествами: Как уже упоминалось, множества могут быть полезными при работе с операциями объединения, пересечения и другими, например, при сравнении списков уникальных пользователей двух различных систем.
3. Применение в алгоритмах: Множества часто используются в различных алгоритмах для учета уникальности элементов или для быстрой проверки принадлежности.

**Словари (dict)**

1. Сөздүк-Бул ачкыч-маани жуптары түрүндөгү маалыматтардын иретсиз жыйнагы.

empty\_dict = {} # Пустой словарь

person = {

"name": "Alex",

"age": 30,

"is\_student": False

}

2. Для доступа к значению по ключу используется следующий синтаксис:

* name = person["name"] # Alex
* age = person.get("age", 25) # 30. Если ключ "age" отсутствует, будет возвращено значение 25.

3. Изменение и добавление элементов

* person["name"] = "Bob" # Изменяет значение для ключа "name"
* person["country"] = "Russia" # Добавляет новую пару ключ-значение

4. Удаление элементов

* del person["age"] # Удаляет ключ "age" и его значение
* country = person.pop("country", "USA") # Удаляет ключ "country" и возвращает его значение. Если ключа нет, возвращает "USA".

Методы словаря

5. keys(): возвращает список всех ключей словаря.

* keys = person.keys()

6. values(): возвращает список всех значений словаря.

* values = person.values()

7. clear(): очищает словарь.

* person.clear()

8. copy(): создает копию словаря.

* person\_copy = person.copy()

9. Вложенные словари

Можно создать словарь, содержащий другие словари:

team = {

"player1": {"name": "John", "position": "forward"},

"player2": {"name": "Alex", "position": "goalkeeper"}

}

10. Применение словарей в бэкенд-разработке

1. Хранение и обработка данных: Словари часто используются для хранения и обработки данных, например, при работе с JSON.
2. Кэширование: Словари могут быть использованы для кэширования результатов, чтобы избежать дорогостоящих операций.
3. Настройки и конфигурация: Многие системы используют словари для хранения настроек и конфигураций.
4. Маршрутизация в веб-фреймворках: Во многих веб-фреймворках маршруты связываются со специфичными функциями или классами с помощью словарей.