ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

@ZTUEDUUA

АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРИЗОВАНИХ ІНФОРМАЦІЙНО-ВИМІРЮВАЛЬНИХ СИСТЕМ

Лекція 9
Тема: Словники

1. Словники
2. Хешовані типи даних
3. Методи словників

1. Словники

Словник в Python — це впорядкована колекція елементів, яка зберігає їх у вигляді пари “ключ-значення”. Ключ — це унікальний ідентифікатор, який пов’язаний із кожним значенням. Наприклад, якщо нам потрібно зберегти інформацію про країни та їх столиці, ми можемо створити словник з назвами країн як ключі та з назвами столиць як значеннями.

Створення словника в Python

Ось як просто можна створити словник в Python:

main.py +

```
1 capital_city = {"Nepal": "Kathmandu", "Ukraine": "Kyiv", "Italy": "Rome"}
2 print(capital_city)
```

Результат:

Ми створили словник **capital_city** , в якому:

Ключі – **«Nepal»** , **«Ukraine»** , **«Italy»**.

Значення – **«Kathmandu»** , **«Kyiv»** , **«Rome»**.

Додавання елементів до словника в Python

Ми можемо додавати елементи до словника, використовуючи ім’я словника з **[ ]** . Наприклад:

main.py +

```
1 capital_city = {"Nepal": "Kathmandu", "Italy": "Rome"}
2 print("Initial Dictionary: ",capital_city)
3
4 capital_city["Japan"] = "Tokyo"
5
6 print("Updated Dictionary: ",capital_city)
```

Результат:

Initial Dictionary: {'Nepal': 'Kathmandu', 'Italy': 'Rome’}

Updated Dictionary: {'Nepal': 'Kathmandu', 'Italy': 'Rome', 'Japan': 'Tokyo’}

Зверніть увагу на рядок коду:

capital_city["Japan"] = "Tokyo"

Тут ми додали новий елемент у словник **capital_city** ключ **Japan**
зі значенням **Tokyo** .

Зміна значень у словнику в Python

Ми також можемо використати [ ] для зміни значення, пов’язаного з певним ключем. Наприклад:

main.py +

```
1 student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
2 print("Initial Dictionary: ", student_id)
3
4 student_id[112] = "Stan"
5
6 print("Updated Dictionary: ", student_id)
```

Результат:

Initial Dictionary: {111: 'Eric', 112: 'Kyle', 113: 'Butters’}

Updated Dictionary: {111: 'Eric', 112: 'Stan', 113: 'Butters’}

Тут ми створили новий словник **student_id** . Спочатку з ключем **112** пов’язане значення **«Kyle»** . Тепер зверніть увагу на наступний рядок коду:

student_id[112] = "Stan"

Тут ми змінили значення, пов’язане з ключем **112** , на **‘Stan’**

Видалення елементів зі словника в Python

В Python **оператор del** використовується для видалення елемента зі словника. Наприклад:

main.py

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}

print("Initial Dictionary: ", student_id)

del student_id[111]

print("Updated Dictionary ", student_id)
```

Результат:

Initial Dictionary: {111: 'Eric', 112: 'Kyle', 113: 'Butters’}

Updated Dictionary {112: 'Kyle', 113: 'Butters’}

Ми також можемо видалити весь словник за допомогою оператора **del**

main.py

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}

del student_id

print(student_id)

# Результат: NameError: name 'student_id' is not defined
```

2. Хешовані типи даних

Слово ***хеш*** походить від англійського «hash», яке можна перекласти як "плутанина", "мішанина", "фарш". Часто ще про сам процес говорять «хешування», від англійського "hashing" (рубати, подрібнювати).

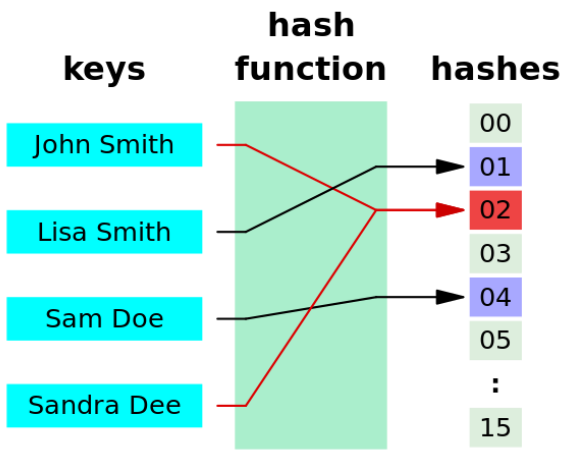
З'явився цей термін в середині минулого століття серед людей, які займались обробкою масивів даних. Спеціальна хеш-функція дозволяла привести будь-який масив даних до числа заданої довжини. Наприклад, якщо байти з послідовності байт будь-якої довжини скласти, і від отриманої суми взяти залишок від ділення на 256, то цей залишок від ділення можна буде називати хешем. Для різних початкових байт залишок від ділення буде відрізнятись.

Але для різної послідовності байт по вищенаведеному алгоритму ми можемо отримати однаковий результат. Наприклад, якщо ми в результаті роботи хеш-функції отримали 1, то початкова послідовність довжиною 2 байти може бути одною з наступних:
0001, 0100, 02FF, 03FE, 04FD, ... FE03, FF02

Тобто для послідовності з 2 байт результат роботи хеш-функції буде однаковим у 256 випадках. І нескладним і відносно невитратним перебором можна отримати усі послідовності з 2 байт від яких хеш-функція видає однаковий результат.
Ситуації, коли хеш-функція видає однаковий результат для різних вхідних даних, називають ***колізіями***.

Вимоги до хеш функцій

Хеш-функція — функція, що перетворює вхідні дані будь-якого (як правило великого) розміру в дані фіксованого розміру.



1. функція повинна вміти приводити будь-який об'єм цифрових даних до числа заданої довжини (по суті — стискання даних до бітової послідовності заданої довжини хитрим способом).
2. найменша зміна (хоча б на один біт) вхідних даних має призводити до повної зміни хешу.
3. функція повинна бути стійкою у зворотній операції — ймовірність відновлення початкових даних по хешу повинна бути досить низькою.
4. функція повинна мати якомога нижчу ймовірність виникнення колізій.
5. хороша хеш-функція не повинна сильно навантажувати обчислювальні потужності. Тут часто важливим є компроміс між швидкістю роботи і якістю результату. Але існують випадки, де від хеш-функції якраз і "вимагається" складність і ресурсозатратність.
6. алгоритм роботи функції має бути відкритим, щоб кожен бажаючий міг би оцінити її стійкість, тобто ймовірність відновлення початкових даних по результату її роботи.

Функція hash() в Python

У цьому прикладі функція **hash()** обчислює хеш-значення рядка **«Hello, World!»** , і виводить це значення. Значення є цілим числом.

```
main.py +
1 text = "Hello, World!"
2 text_hash = hash(text)
3 print(text_hash)
```

Результат:

-7153501429386363 00

У цьому прикладі ми створюємо словник person, а потім хешуємо його, використовуючи **‘frozenset’** з пар ключ-значення методу **‘items()’** . Це дозволяє створити незмінний об'єкт, який можна хешувати.

```
main.py +
1 person = {"name": "John", "age": 30}
2 person_hash = hash(frozenset(person.items()))
3 print(person_hash)
```

Результат:

8933660234574609194

‘frozenset’ в Python - це незмінний тип даних, який подібний до звичайної множини (**set**) .

3. Методи словників

1. Вбудований метод **iter()** дозволяє повернути ітератор за ключами словника. Ітератори забезпечують покроковий обхід списку ключів.

```
main.py +
1  my_dict = {'a': 1, 'b': 2, 'c': 3}
2  items_iterator = iter(my_dict.items())
3  for key, value in items_iterator:
4      print(key, value)
```

Результат:
a 1
b 2
c 3

2. Метод **clear()** видаляє усі елементи зі словника.

```
main.py +
1  my_dict = {'a': 1, 'b': 2, 'c': 3}
2
3  print("Словник до очищення:", my_dict)
4
5  my_dict.clear()
6
7  print("Порожній словник після очищення:", my_dict)
```

Результат:

Словник до очищення: {'a': 1, 'b': 2, 'c': 3}

Порожній словник після очищення: {}

3. У випадку, якщо потрібно створити копію зі словника, використовується метод **copy()**.

```
main.py +
1  original_dict = {'a': 1, 'b': 2, 'c': 3}
2
3  copied_dict = original_dict.copy()
4
5  original_dict['a'] = 10
6
7  print("Оригінальний словник:", original_dict)
8  print("Скопійований словник:", copied_dict)
```

Результат:
Оригінальний словник: {'a': 10, 'b': 2, 'c': 3}

Скопійований словник: {'a': 1, 'b': 2, 'c': 3}

3. Методи словників

4. Метод **fromkeys()** класу **dict** дозволяє створити новий словник з послідовності заданих ключів та деякого значення.

```
main.py
keys = ['a', 'b', 'c']
default_value = 0

my_dict = dict.fromkeys(keys, default_value)

print(my_dict)
```

Результат:

{'a': 0, 'b': 0, 'c': 0}

5. Метод **get()** призначений для отримання значення за заданим ключем якщо ключ є в словнику. Якщо ключа немає в словнику, то повертається значення встановлене за замовчуванням.

```
main.py
my_dict = {'a': 1, 'b': 2, 'c': 3}

value_a = my_dict.get('a')
value_d = my_dict.get('d')

print("Значення для ключа 'a':", value_a)
print("Значення для ключа 'd':", value_d)
```

Результат:
Значення для ключа 'a': 1

Значення для ключа 'd': None

6. Метод **items()** повертає нове представлення елементів словника у вигляді пар ключ: значення.

```
main.py
my_dict = {'a': 1, 'b': 2, 'c': 3}

dict_items = my_dict.items()

for item in dict_items:
    print(item)
```

Результат:

('a', 1)
('b', 2)
('c', 3)

3. Методи словників

7. Метод **keys()** дозволяє отримати список ключів.

main.py

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

keys = my_dict.keys()

for key in keys:
    print(key)
```

Результат:

a
b
c

8. Метод **pop()** використовується для видалення елементу зі словника за заданим ключом.

main.py

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

value_b = my_dict.pop('b')

print("Значення для ключа 'b':", value_b)
print("Словник після вилучення:", my_dict)
```

Результат:

Значення для ключа 'b': 2

Словник після вилучення: {'a': 1, 'c': 3}

9. Метод **popitem()** видаляє і повертає випадкову пару (ключ, значення) зі словника. Метод використовується для деструктивного перебору словника, який часто використовується в алгоритмах множин.

main.py

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

last_item = my_dict.popitem()

print("Остання пара ключ-значення:", last_item)
print("Словник після вилучення:", my_dict)
```

Результат:
Остання пара ключ-значення: ('c', 3)
Словник після вилучення: {'a': 1, 'b': 2}

3. Методи словників

10. Метод **setdefault()** дозволяє за заданим ключем встановити елемент за замовчуванням.

main.py

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

value_b = my_dict.setdefault('b', 10)  # Ключ 'b' вже існує

value_d = my_dict.setdefault('d', 10)  # Ключ 'd' відсутній

print("Значення для ключа 'b':", value_b)  # Виведе поточне значення для 'b'
print("Значення для ключа 'd':", value_d)  # Виведе значення за замовчуванням
print("Словник після встановлення значення для ключа 'd':", my_dict)
```

Результат:
Значення для ключа 'b': 2
Значення для ключа 'd': 10
Словник після встановлення значення для ключа 'd': {'a': 1, 'b': 2, 'c': 3, 'd': 10}

11. Метод **update()** використовується для оновлення словнику на основі заданого списку пар (**ключ:значення**). Метод повертає **None**.

main.py

```
my_dict = {'a': 1, 'b': 2, 'c': 3}
other_dict = {'b': 4, 'd': 5, 'e': 6}

my_dict.update(other_dict)

print("Оновлений словник my_dict:", my_dict)
```

Результат:
Оновлений словник my_dict: {'a': 1, 'b': 4, 'c': 3, 'd': 5, 'e': 6}

12. Метод **values()** дозволяє отримати список значень з заданого словника.

main.py

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

values = my_dict.values()

for value in values:
    print(value)
```

Результат:

1
2
3

@ZTUEDUUA

ЖИТОМИРСЬКА ПОЛІТЕХНІКА

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

- **Розвиваємо лідерів**
- **Створюємо інновації**
- **Змінюємо світ на краще**