

Python

Словари (продолжение)



Преподаватель

Портрет

Имя Фамилия

Текущая должность

Количество лет опыта

Какой у Вас опыт - ключевые кейсы

Самые яркие проекты

Дополнительная информация по вашему усмотрению

Корпоративный e-mail

Социальные сети (по желанию)

Важно

-  Камера должна быть включена на протяжении всего занятия
-  В течение занятия вопросы задавать в чате или когда преподаватель спрашивает, есть ли у Вас вопросы
-  Вести себя уважительно и этично по отношению к остальным участникам занятия
-  Организационные вопросы по обучению решаются с кураторами, а не на тематических занятиях
-  Во время занятия будут интерактивные задания, будьте готовы включить камеру или демонстрацию экрана по просьбе преподавателя

План занятия

- Метод fromkeys
- Метод get
- Метод setdefault
- Методы keys, values, items
- Словари со вложенными структурами



ОСНОВНОЙ БЛОК





Метод fromkeys



Метод fromkeys

Этот метод позволяет создавать словарь, где указанным ключам присваивается одинаковое значение.

Синтаксис



```
dict.fromkeys(iterable, value)
```


Примеры



Создание словаря без указания значения

```
keys = ["x", "y", "z"]
```

```
my_dict = dict.fromkeys(keys) # Каждому ключу присваивается значение `None`
```

```
print(my_dict) # {'x': None, 'y': None, 'z': None}
```

Создание словаря с переданным значением

```
keys = [1, 2, 3]
```

```
default_value = "default"
```

```
my_dict = dict.fromkeys(keys, default_value)
```

```
print(my_dict) # {1: 'default', 2: 'default', 3: 'default'}
```

Создание словаря с изменяемыми значениями



Будьте осторожны при использовании изменяемых значений (например, списка), так как все ключи будут ссылаться на один и тот же объект.

Создание словаря с изменяемыми значениями



```
keys = ["a", "b", "c"]

shared_list = []

my_dict = dict.fromkeys(keys, shared_list)

my_dict["a"].append(1) # Изменит общий список для всех ключей

print(my_dict) # {'a': [1], 'b': [1], 'c': [1]}
```



ВОПРОСЫ





Метод get



Метод `get()`

Этот метод используется для безопасного доступа к значению по указанному ключу. В отличие от `dictionary[key]`, метод `get()` не вызывает ошибку `KeyError`, если ключ отсутствует, а вместо этого возвращает значение по умолчанию.

Синтаксис



```
value = dictionary.get(key, default_value)
```

Метод get



dictionary — словарь, из которого нужно получить значение.



key — ключ, по которому производится поиск значения.



default_value (необязательный параметр) — значение, которое будет возвращено, если ключ отсутствует. Если не указано, возвращается **None**.

Примеры



Получение значения по существующему ключу

```
my_dict = {"name": "Alice", "age": 30}
```

```
print(my_dict.get("name")) # Alice
```

```
print(my_dict.get("name", "Anonim")) # Alice
```

Запрос отсутствующего ключа без указания значения по умолчанию

```
print(my_dict.get("city")) # None
```

Запрос отсутствующего ключа с указанным значением по умолчанию

```
print(my_dict.get("city", "Unknown")) # Unknown
```



ВОПРОСЫ





Метод
setdefault



Метод setdefault

Этот метод используется для получения значения по указанному ключу. Если ключ отсутствует в словаре, метод добавляет его с указанным значением по умолчанию и возвращает это значение. Если ключ уже существует, `setdefault()` просто возвращает его текущее значение без изменений.

Синтаксис



```
value = dictionary.setdefault(key, default_value)
```

Метод get



dictionary — объект словаря, из которого вы хотите получить значение.



key — ключ, который нужно найти или добавить.



default_value (необязательный параметр) — значение, которое будет добавлено, если ключ отсутствует. Если не указано, используется **None**.

Примеры



```
# Получение значения существующего ключа

my_dict = {"name": "Alice", "age": 30}

age = my_dict.setdefault("age")

print(age) # 30

print(my_dict) # {'name': 'Alice', 'age': 30}
```

Примеры



```
# Добавление нового ключа без указания значения по умолчанию  
  
country = my_dict.setdefault("country")  
  
print(country) # None  
  
print(my_dict) # {'name': 'Alice', 'age': 30, 'country': None}
```


Примеры



```
# Добавление нового ключа с переданным значением

city = my_dict.setdefault("city", "Unknown")

print(city) # Unknown

print(my_dict) # {'name': 'Alice', 'age': 30, 'country': None, 'city': 'Unknown'}
```

Преимущества метода `setdefault()`



Упрощает работу со словарями, когда необходимо гарантировать наличие ключа с определённым значением. Позволяет избежать дополнительных проверок `if key in dictionary` перед добавлением нового ключа.



ВОПРОСЫ





ЗАДАНИЕ





Выберите верный вариант ответа

Что произойдёт, если вызвать метод `get()` для отсутствующего ключа без указания значения по умолчанию?

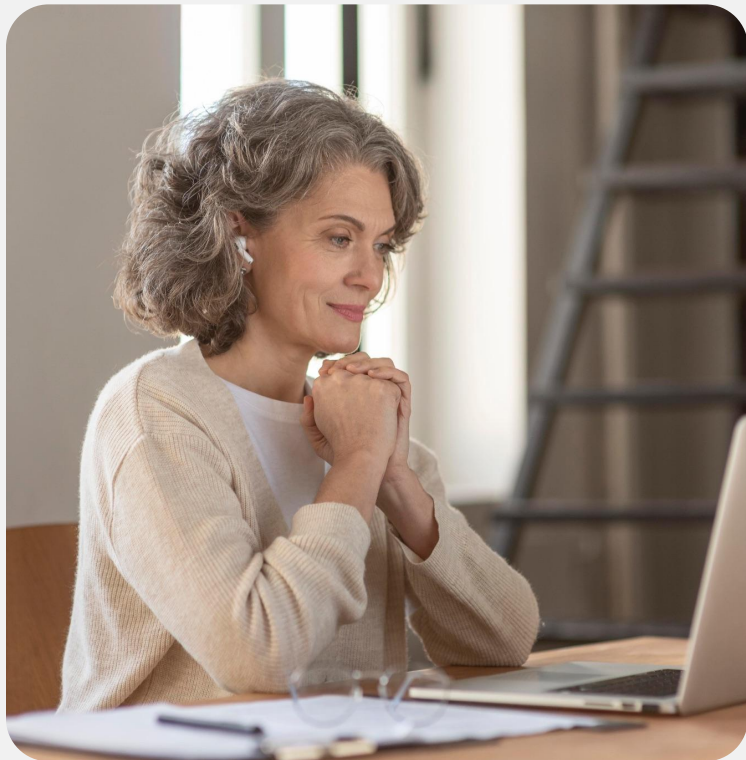
```
my_dict = {"name": "Alice", "age": 30}
```

```
value = my_dict.get("city")
```

```
print(value)
```

- a. Вернётся значение Unknown
- b. Вернётся значение None
- c. Вернётся пустая строка ""
- d. Возникнет ошибка

KeyError



Выберите верный вариант ответа

Что произойдёт, если вызвать метод `get()` для отсутствующего ключа без указания значения по умолчанию?

```
my_dict = {"name": "Alice", "age": 30}
```

```
value = my_dict.get("city")
```

```
print(value)
```

- a. Вернётся значение Unknown
- b. Вернётся значение None**
- c. Вернётся пустая строка ""
- d. Возникнет ошибка

KeyError



Выберите верный вариант ответа

Что произойдёт, если ключ уже существует при использовании метода `setdefault()`?

```
my_dict = {"name": "Alice", "age": 30}
```

```
result = my_dict.setdefault("age", 25)
```

```
print(result)
```

```
print(my_dict)
```

- Добавится новый ключ "age": 25
- Вернётся значение 30, словарь останется неизменным
- Вернётся значение 30, словарь будет дополнен ключом "age": 25
- Ошибка



Выберите верный вариант ответа

Что произойдёт, если ключ уже существует при использовании метода `setdefault()`?

```
my_dict = {"name": "Alice", "age": 30}
```

```
result = my_dict.setdefault("age", 25)
```

```
print(result)
```

```
print(my_dict)
```

- a. Добавится новый ключ "age": 25
- b. Вернётся значение 30, словарь останется неизменным**
- c. Вернётся значение 30, словарь будет дополнен ключом "age": 25
- d. Ошибка



ВОПРОСЫ





Методы keys,
values, items

Методы keys, values, items



Словари в Python обладают тремя полезными методами для работы с ключами, значениями и парами "ключ-значение". Эти методы возвращают динамические представления (*view objects*), которые ссылаются на словарь и автоматически отражают изменения в его содержимом.

Метод `keys()`



Представление обновляется автоматически при изменении словаря.



Можно преобразовать результат в список или другую коллекцию для получения копии элементов.

Пример



```
my_dict = {"name": "Alice", "age": 30}
keys = my_dict.keys()
print(type(keys))
print(keys)
```

```
# Изменение словаря
my_dict["city"] = "New York"
print(keys) # Значения обновляются
```

```
# Преобразование представления в список
keys_list = list(keys)
my_dict["country"] = "USA"
print(keys_list) # На списке изменения не отображаются
```

Метод values()



Представление автоматически обновляется при изменении словаря.



Можно преобразовать результат в список или другую коллекцию для получения копии элементов.

Пример



```
my_dict = {"name": "Alice", "age": 30}
values = my_dict.values()
print(type(values))
print(values)
```

```
# Изменение словаря
my_dict["age"] = 31
print(values) # Значение обновляется
```

```
# Преобразование представления в список
values_list = list(values)
my_dict["age"] = 32
print(values_list) # Изменения на списке не отображаются
```

Метод items()



Возвращает представление всех пар "ключ-значение" в словаре в виде кортежей (ключ, значение).

Метод items()



Представление автоматически обновляется при изменении словаря.



Можно преобразовать результат в список (с кортежами) или другую коллекцию для получения копии элементов.

Цикл по словарю



Словари позволяют итерироваться по своим ключам, значениям или парам "ключ-значение" с помощью цикла `for`.

Итерация по ключам



```
my_dict = {"name": "Alice", "age": 30, "city": "New York"}
```

```
for key in my_dict:
```

```
    print(key)
```

```
# Аналогично циклу по my_dict
```

```
for key in my_dict.keys():
```

```
    print(key)
```

Итерация по значениям



Для перебора значений используется метод `values()`.

Итерация по ключам



```
my_dict = {"name": "Alice", "age": 30, "city": "New York"}  
  
for value in my_dict.values():  
    print(value)
```

Итерация по парам "ключ-значение"



Для перебора пар "ключ-значение" используется метод `items()`.

Итерация по парам "ключ-значение"



```
my_dict = {"name": "Alice", "age": 30, "city": "New York"}  
  
for key, value in my_dict.items():  
    print(f"{key}: {value}")
```



ВОПРОСЫ





ЗАДАНИЕ





Выберите правильный вариант ответа

Какой тип данных возвращает метод `items()`?

```
my_dict      =      {"x":      10,      "y":      20}
items        =      my_dict.items()
print(type(items))
```

- a. `list`
- b. `set`
- c. `tuple`
- d. `dict_items`



Выберите правильный вариант ответа

Какой тип данных возвращает метод `items()`?

```
my_dict      =      {"x":      10,      "y":      20}
items        =      my_dict.items()
print(type(items))
```

- a. `list`
- b. `set`
- c. `tuple`
- d. `dict_items`



ВОПРОСЫ





Словари со вложенными структурами

Словари со вложенными структурами



Словари могут содержать вложенные структуры данных, такие как списки, кортежи, множества и другие словари в качестве значений.

Списки внутри словаря



```
student_scores = {  
    "Alice": [90, 85, 88],  
    "Bob": [72, 75, 80],  
    "Charlie": [95, 100, 98]  
}
```

Вложенные словари и списки



```
school = {
    "class1": {
        "students": ["Alice", "Bob", "Charlie"],
        "teacher": "Mrs. Smith"
    },
    "class2": {
        "students": ["David", "Eva"],
        "teacher": "Mr. Johnson"
    }
}
```


Доступ к элементам в словаре со вложенными структурами



Для доступа к элементам необходимо указывать ключи и индексы в зависимости от типа вложенной структуры.

Примеры



```
student_scores = {  
    "Alice": [90, 85, 88],  
    "Bob": [72, 75, 80],  
    "Charlie": [95, 100, 98]  
}  
  
# Доступ к элементу списка внутри словаря  
print(student_scores["Alice"][1])
```

Примеры



```
school = {
    "class1": {
        "students": ["Alice", "Bob", "Charlie"],
        "teacher": "Mrs. Smith"
    },
    "class2": {
        "students": ["David", "Eva"],
        "teacher": "Mr. Johnson"
    }
}
```

Доступ к элементу вложенного словаря

```
print(school["class2"]["teacher"])
```

Доступ к элементу списка, полученного из вложенного словаря

```
print(school["class1"]["students"][0])
```

Итерация по словарю со вложенными структурами



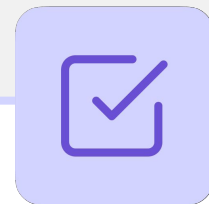
В зависимости от типа вложенной структуры может потребоваться использовать вложенные циклы для перебора элементов.

Пример



```
for class_name, details in school.items():
    print(f"Class: {class_name}")
    for key, value in details.items():
        print(f"\t\t{key}: {value}")
```

Обновление элементов



Можно добавлять, изменять или удалять данные во вложенных структурах, используя доступ по ключам и индексам.

Пример



Добавление ученика в список

```
school["class1"]["students"].append("Daisy")
```

```
print(school["class1"]["students"])
```

Пример



Удаление ключа из вложенного словаря

```
del school["class2"]["teacher"]
```

```
print(school["class2"])
```


Практическое применение



Словари со вложенными структурами удобны для работы с данными, имеющими сложную иерархию.

Например



JSON-файлы



Результаты сложных вычислений



Конфигурации приложений

Копирование словарей



Создаёт новый словарь с копией всех ключей и значений из исходного словаря. Ключи и значения копируются "по ссылке", что означает, что изменения в вложенных объектах (например, списках) будут отражаться и в оригинале.

Пример



```
original_dict = {"name": "Alice", "age": 30, "scores": [90, 85, 88]}

copied_dict = original_dict.copy()

print(copied_dict)

# Изменение не затронет копию для неизменяемых элементов

original_dict["age"] = 31

# Изменение затронет копию для изменяемых элементов

original_dict["scores"].append(80)

print(original_dict)

print(copied_dict)
```

Глубокое копирование с помощью модуля сору



Глубокое копирование создаёт полностью независимую копию словаря, включая все вложенные объекты. Изменения во вложенных объектах копии не будут затрагивать оригинал и наоборот.

Использование функции `deepcopy()`:



```
import copy

original_dict = {"name": "Alice", "age": 30, "scores": [90, 85, 88]}

deep_copied_dict = copy.deepcopy(original_dict)

print(deep_copied_dict)

# Любые изменения во вложенном объекте не затронут копию

original_dict["age"] = 31

original_dict["scores"].append(80)

print(original_dict)

print(deep_copied_dict)
```

Dict comprehension



Dict comprehension позволяет создавать словари с использованием краткой и удобной конструкции, подобно генераторам списков. Это делает код более лаконичным и читаемым, особенно при создании или преобразовании словарей из других итерируемых объектов.

Синтаксис



```
new_dict = {key_expr: value_expr for item in iterable}
```


Например



key_expr — выражение для формирования ключа.



value_expr — выражение для формирования значения.



iterable — любой итерируемый объект (список, диапазон и т. д.).

Примеры использования



```
# Создание словаря с квадратами чисел из списка
numbers = [1, 2, 3, 4]
squared_dict = {num: num ** 2 for num in numbers}
print(squared_dict)
```

```
# Фильтрация элементов
original_dict = {"a": 5, "b": 2, "c": 0, "d": 3, "e": 0, "f": 3}
filtered_dict = {key: value for key, value in original_dict.items() if value > 0}
print(filtered_dict)
```

```
# Анализ данных и сохранение результатов
words = ["apple", "banana", "cherry"]
length_dict = {word: len(word) for word in words}
print(length_dict)
```

Сравнение словарей



В Python можно сравнивать словари по их содержимому, используя стандартные операторы сравнения. Сравнение словарей происходит на основании ключей и их соответствующих значений.

Сравнение на равенство (==)



Два словаря считаются равными, если они содержат одинаковые пары "ключ-значение", независимо от порядка их добавления.

Синтаксис



```
dict1 = {"a": 1, "b": 2}
dict2 = {"b": 2, "a": 1}
print(dict1 == dict2) # True
```

Сравнение на равенство (==)



Если хотя бы одна пара "ключ-значение" отличается, словари считаются неравными.

Синтаксис



```
dict1 = {"a": 1, "b": 2}
dict2 = {"a": 1, "b": 2, "c": 3}
print(dict1 == dict2) # False
```

Сравнение на неравенство (!=)



Если словари имеют разные пары "ключ-значение", они считаются неравными.

Синтаксис



```
dict1 = {"a": 1, "c": 1, "b": [2, 1, 5]}  
dict2 = {"b": [2, 1, 5], "a": 1, "c": 1}  
print(dict1 != dict2) # False
```

Особенности сравнения словарей



Порядок элементов не влияет на результат сравнения.



Если словари содержат изменяемые объекты (например, списки), то при сравнении учитываются их значения, а не ссылки на объекты.



Доступны только операции сравнения на равенство (==) и неравенство (!=).



ВОПРОСЫ





ЗАДАНИЕ



Какой результат будет выведен при выполнении следующего кода?

```
company = {
    "department1": {
        "employees": ["John", "Doe"],
        "manager": "Mr. Anderson"
    },
    "department2": {
        "employees": ["Jane", "Smith"],
        "manager": "Mrs. Carter"
    }
}

company["department2"]["employees"].append("Miller")
print(company["department2"]["employees"])
```

a. ["John", "Doe", "Miller"]
 b. ["Jane", "Smith"]
 c. ["Jane", "Smith", "Miller"]
 d. ["Jane", "Smith", "John", "Doe", "Miller"]
 e. Ошибка

Какой результат будет выведен при выполнении следующего кода?

```
company = {
    "department1": {
        "employees": ["John", "Doe"],
        "manager": "Mr. Anderson"
    },
    "department2": {
        "employees": ["Jane", "Smith"],
        "manager": "Mrs. Carter"
    }
}

company["department2"]["employees"].append("Miller")
print(company["department2"]["employees"])
```

a. ["John", "Doe", "Miller"]
 b. ["Jane", "Smith"]
 c. **["Jane", "Smith", "Miller"]**
 d. ["Jane", "Smith", "John", "Doe", "Miller"]
 e. Ошибка



ВОПРОСЫ





ПРАКТИЧЕСКИЕ ЗАДАНИЯ





Переводчик

Напишите программу, которая позволяет пользователю переводить слова между английским и русским языками. Пользователь вводит слово, программа ищет его перевод в словаре. Если слово отсутствует, программа выводит сообщение об отсутствии перевода.

Данные:

```
dictionary = {  
    "Butterfly": "Бабочка",  
    "Training": "Обучение",  
    "Restaurant": "Ресторан",  
    "Programming": "Программирование",  
}
```



Пример вывода

Введите слово для перевода (или 'exit' для выхода):
Бабочка

Перевод: Butterfly

Введите слово для перевода (или 'exit' для выхода):
Butterfly

Перевод: Бабочка

Введите слово для перевода (или 'exit' для выхода): Travel

Перевод отсутствует.

Введите слово для перевода (или 'exit' для выхода): exit

Программа завершена.



Решение

```
dictionary = {  
    "Butterfly": "Бабочка",  
    "Training": "Обучение",  
    "Restaurant": "Ресторан",  
    "Programming": "Программирование",  
}
```



Решение

```
while True:
```

```
    # Ввод слова
```

```
    word = input("Введите слово для перевода (или 'exit'  
для выхода): ").capitalize()
```

```
    if word == "Exit":
```

```
        print("Программа завершена.")
```

```
        break
```



Решение

Поиск перевода в прямом и обратном словарях

```
if word in dictionary:
    print(f"Перевод: {dictionary[word]}")
elif word in dictionary.values():
    for key, value in dictionary.items():
        if value == word:
            print(f"Перевод: {key}")
            break
else:
    print("Перевод отсутствует.")
```



Правильность расстановки скобок

Напишите программу, которая принимает строку, содержащую любые виды скобок — круглые `()`, квадратные `[]`, и фигурные `{}`. Необходимо проверить, правильно ли они расставлены. Скобки считаются правильно расставленными, если:

- Каждая открывающая скобка имеет соответствующую закрывающую.
- Скобки закрываются в правильном порядке.
- Подумайте, как вам помогут словари в решении.



Правильность расстановки скобок

Данные:

```
string = "([])"]
```

```
# string = ({[]})
```

Пример вывода:

```
Скобки '([])'] валидны: False
```

```
Скобки '({[]})' валидны: True
```



Решение

```
string = "([)]"

is_brackets_valid = True

stack = [] # Используем список как стек

brackets = {')': '(', ']': '[', '}': '{'} # Сопоставление
скобок
```




Решение

```
for char in string:
    if char in brackets.values(): # Если открывающая скобка
        stack.append(char)
    elif char in brackets.keys(): # Если закрывающая скобка
        if stack and stack[-1] == brackets[char]:
            stack.pop() # Удаляем последнюю открывающую
скобку
        else:
            is_brackets_valid = False # Если не совпадает,
порядок нарушен
print(f"Скобки '{string}' валидны:", is_brackets_valid)
```



ДОМАШНЕЕ ЗАДАНИЕ



Домашнее задание

Реверс словаря

Напишите программу, которая меняет местами ключи и значения в словаре. Если значения повторяются, добавьте их в список.

Данные:

```
data = {"a": 1, "b": 2, "c": 1, "d": 3}
```

Пример вывода:

Перевернутый словарь: {1: ['a', 'c'], 2: ['b'], 3: ['d']}

Домашнее задание

Счётчик букв в словах

Напишите программу, которая подсчитывает количество каждой буквы в заданных словах и создаёт словарь, где ключи — это слова, а значения — это ещё один словарь с подсчётом каждой буквы.

Данные:

```
words = ["anna", "bennet", "john"]
```

Пример вывода:

```
{'anna': {'a': 2, 'n': 2}, 'bennet': {'b': 1, 'e': 2, 'n': 2, 't': 1}, 'john': {'j': 1, 'o': 1, 'h': 1, 'n': 1}}
```

Домашнее задание

Распределение студентов по группам

У вас есть словарь, где ключи — имена студентов, а значения — их баллы за экзамен. Необходимо распределить студентов по группам:

- **"Отличники"**: баллы ≥ 85
- **"Хорошисты"**: баллы от 70 до 84
- **"Троечники"**: баллы от 50 до 69
- **"Не сдали"**: баллы < 50

Создайте словарь с ключами-группами и словарями со студентами в качестве значений.

Данные:

```
students = {"Аня": 92, "Боря": 76, "Ваня": 65, "Галя": 48, "Дима": 88, "Ева": 54}  
groups = ["Отличники", "Хорошисты", "Троечники", "Не сдали"]
```

Заключение

