Лекція 15
Словники

Контрольна робота №5
Правила визначення варіанту

IO	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
61	29	1	2	3	4	5	6	7	8	9	10	11	12	13	14
62	27	28	29	1	2	3	4	5	6	7	8	9	10	11	12
63	25	26	27	28	29	1	2	3	4	5	6	7	8	9	10
64	23	24	25	26	27	28		1	2	3	4	5	6	7	8
65	21	22	23	24	25	26	27			1	2	3	4	5	6

Червоний колір – номер варіанту **Чорний та синій колір – номер у списку**

IO	16	17	18	19	20	21	22	23	24	25	26	27	28	29
61	15	16	17	18	19	20	21	22	23	24	25	26	27	28
62	13	14	15	16	17	18	19	20	21	22	23	24	25	26
63	11	12	13	14	15	16	17	18	19	20	21	22	23	24
64	9	10	11	12	13	14	15	16	17	18	19	20	21	22
65	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Записати результат роботи програми.
За неможливості передбачення точної відповіді записати ДІЮ інструкції, наприклад: «вибирає елемент випадковим чином»

1.	>>> arr = [1, 2, 1, 2, 1, 7] >>> arr.index(1), arr.index(2)	16.	>>> arr = [1, 2, 1, 2, 1, 7] >>> arr.index(1, 1), arr.index(1, 3, 5)
2.	>>> arr = [1, 2, 1, 2, 1, 7] >>> arr.index(3)	17.	>>> arr = [1, 2, 1, 2, 1, 7] >>> arr.index(5)
3.	>>> arr = [1, 2, 1, 2, 1, 2, 3, 2] >>> arr.count(1), arr.count(2)	18.	>>> arr = [1, 2, 1, 2, 1] >>> arr.count(4), arr.count(1)
4.	>>> 2 in [1, 2, 3, 4, 5]	19.	>>> 6 not in [1,2,3,4,5]
5.	>>> any([0, None, 1])	20.	>>> any([0, None])
6.	>>> all([18, "None"])	21.	>>> all([])
7.	>>> arr = [1, 20, 3, 40, 5] >>> arr.reverse(); arr	22.	arr = [10, 2, 40, 5, 80, 9] print([i for i in reversed(arr)])
8.	arr = [10, 2, 40, 5, 80, 9] print(list(reversed(arr)))	23.	>>> import random >>> random.choice((1, 1.12345, "t"))

9.	>>> t = (1, 2, 3); t[0]=50; t	24.	>>> t = (1, 2, 3); t.index(1)
10.	>>> t = (2, 2, 3, 2, 9); t.count(2)	25.	>>> set([1, 2, 3, 1, 2, 3])

11.	>>> import random
	>>> arr = [1, 2, 7, 8, 9, 10]
	>>> random.shuffle(arr)
		26.	def getkey(item): return item[1]
			s = [[9, 3], [1, 7],[3, 64]]
			s.sort(key=getkey); print(s)

12.	>>> arr = [3, 4, 5, 6, 7, 8, 9, 10] >>> random.sample(arr, 4)	27.	>>> arr = [2, 7, 1, 4] >>> arr.sort(); arr
13.	>>> arr = [2, 7, 1, 4] >>> arr.sort(reverse = True); arr	28.	a = ["c", "a", "C", "b", "B"] a1=sorted(a, key=str.lower); print(a1)
14.	arr = ["c", "a", "C", "b", "B"] arr.sort(key=str.lower); print(arr)	29.	>>> arr = ["Fac", "book", "r"] >>> "e".join((str(i) for i in arr))
15.	>>> t = (1, 2, 3, 4, 5, 6, 7, 8, 9) >>> t[2:5]	30.	def getkey(item): return item[0] s = [[9, 3], [1, 7], [3, 64]] s.sort(key=getkey, reverse=True) print(s)

Основні характеристики словників

Словники – це набори об'єктів, доступ до яких здійснюється не по індексу, а по ключу.

1. Кожний елемент словника містить пари:

<ключ>:<значення>

2. Ключ - це незмінюваний об'єкт:

число, рядок або кортеж.

3. Елементи словника можуть:

містити об'єкти довільного типу даних

мати необмежений ступінь вкладеності.

4. Елементи в словниках розташовуються в довільному порядку.

5. Щоб одержати елемент, необхідно вказати ключ, який використовувався при збереженні значення.

6. Словники є відображеннями, а не послідовностями. Тому функції списків і кортежів до них незастосовні.

7. Як і списки, словники є змінюваними типами даних. Іншими словами, ми можемо не тільки одержати значення по ключу, але й змінити його.

Створення словника

Існує 4 способи створення словників

Спосіб 1. Створюємо словник за допомогою функції `dict()` .

Формати функції:

dict(<Ключ1>=<Значення1>[,...,<КлючN>=<ЗначенняN>])

dict (<Словник>)

dict(<Список кортежів з двома елементами (Ключ, Значення)>)

dict(<Список списків з двома елементами [Ключ, Значення]>)

Якщо параметри не зазначені, то створюється порожній словник.

Приклад 1. Створення словників у спосіб 1

```
>>> d = dict(); d  # Створюємо порожній словник
{ }

>>> d = dict(a = 1, b = 2); d # ключ=значення
{'b': 2, 'a': 1}

>>> d = dict({"a" : 1, "b" :  2}); d # Словник
{'b': 2, 'a': 1}

# Список кортежів
>>> d = dict ([("a",1), ("b", 2)]); d
{'b': 2, 'a': 1}

# Список списків
>>> d = dict ([["a",1], ["b", 2]]); d
{'b': 2, 'a': 1}
```

Використання функції `zip()`

1. Об'єднуєм два списки в список кортежів за допомогою функцій **`list`**`()` і **`zip`**`()`.

2. Потім цей список можна використовувати для створення словника

Приклад 2.

```
>>> k = ["a", "b"] # Список з ключами
>>> v = [1, 2]     # Список зі значеннями

>>> list(zip(k, v)) # Створення списку кортежів
[('a', 1), ('b', 2)]
d = dict ([("a",1), ("b", 2)])
{'a': 1, 'b': 2}

>>> d = dict(zip(k, v)); d # Створення словника
{'a': 1, 'b': 2}
```

Спосіб 2. Створюємо словник, вказавши всі елементи всередині фігурних дужок.

Це найбільш часто використовуваний спосіб створення словника. Між ключем і значенням вказують двокрапку, а пари «ключ/значення» записують через кому.

Приклад 3.

```
>>> d = {}; d  # Створення порожнього словника
{}
>>> d = {"Ann":19, "Wolf":22 }; d
{'Ann': 19, 'Wolf': 22}

>>> d = {"Пн":1, "Вт":2, "Ср":2, "Чт":2}; d
{'Вт': 2, 'Ср': 2, 'Пн': 1, 'Чт': 2}

>>> d = {("Пн","Вер"):1, ("Вт","Жовт"):2}; d
{('Пн', 'Вер'): 1, ('Вт', 'Жовт'): 2}
```

Спосіб 3. Створюємо словник, заповнивши словник поелементно

У цьому випадку ключ вказується усередині квадратних дужок:

Приклад 4.

```
>>> d = {}       # Створюємо порожній словник
>>> d["a"] = 1 # Додаємо елемент1 (ключ "a")
>>> d["b"] = 2 # Додаємо елемент2 (ключ "b")
>>> d["c"] = 3 # Додаємо елемент3 (ключ "c")
>>> d
{'b': 2, 'a': 1, 'c': 3}

>>> nest = {}
>>> nest["Africa","South"]=1
>>> nest["America","West"]=2
>>> nest
{('America', 'West'): 2, ('Africa', 'South'): 1}
```

Спосіб 4. Створюємо словник, за допомогою методу

dict.fromkeys(<Послідовність>[, <Значення>])

Метод створює новий словник.

1. Ключами словника будуть елементи послідовності, переданої першим параметром
2. Значенням елементів словника буде величина, передана другим параметром.
3. Якщо другий параметр не зазначений, то значенням елементів словника буде значення None.

Приклад 5.

```
# немає другого параметра
>>> d=dict.fromkeys(["winter","summer","spring"])
>>> d
{'winter': None, 'spring': None, 'summer': None}
# Ключ – кортеж, другий параметр 0
>>> d = dict.fromkeys(("Пн", "Вт", "Ср"), 0); d
{'Ср': 0, 'Пн': 0, 'Вт': 0}
```

Групове присвоювання

1. При створенні словника в змінній зберігається посилання на об'єкт, а не сам об'єкт. Це обов'язково слід враховувати при груповому присвоюванні.
2. Групове присвоювання можна використовувати для чисел і рядків, але для списків і словників цього робити не можна.

Приклад 6.

```
# Нібито створили два об'єкти
>>> d1 = d2 = { "a": 1, "b": 2}
>>> d2["b"] = 10
>>> d1, d2 # Змінилося значення у двох змінних
({'a': 1, 'b': 10}, {'a': 1, 'b': 10})
```

Як видно з прикладу, зміна значення в змінній `d2` привела також до зміни значення в змінній `d1`. Тобто, обидві змінні посилаються на той самий об'єкт, а не на два різних об'єкти.

Позиційне присвоювання для словників

Щоб одержати два об'єкти, виконуємо позиційне присвоювання.

Приклад 7.

```
>>> d1, d2 = {"a": 1, "b": 2}, {"a": 1, "b": 2}
>>> d2["b"] = 10
>>> d1, d2
({'a': 1, 'b': 2}, {'a': 1, 'b': 10})
```

```
group1={"John": 23, "Mary": 18}
group2={"John": 23, "Mary": 18}
group1["John"]=40
print(group1,group2)
Результат роботи:
{'John': 40, 'Mary': 18} {'John': 23, 'Mary': 18}
```

Створити поверхневу копію словника функцією `dict()`

Формат: `<Словник2> = dict(<Словник1>)`

Приклад 8.

```
>>> d1 = {"a": 1, "b": 2} # Створюємо словник
>>> d2 = dict(d1) # Створюємо поверхневу копию
>>> d1 is d2 # це різні об'єкти
False

>>> d2["b"] = 10
>>> d1, d2 # Змінилася тільки змінна d2
({'a': 1, 'b': 2}, {'a': 1, 'b': 10})
```

Створити поверхневу копію словника методом `copy()`

Приклад 9.

```
>>> d1 = {"a": 1, "b": 2} # Створюємо словник

>>> d2 = d1.copy() # Створюємо поверхневу копию

>>> d1 is d2 # Оператор показує, що це різні об'єкти
False

>>> d2 ["b"] = 10
>>> d1, d2 # Змінилося тільки значення в змінній
d2
({'a': 1, 'b': 2}, {'a': 1, 'b': 10})
```

Створити глибоку копію словника функцією `deepcopy()` з модуля `copy`

Приклад 10.

```
>>> d1 = {"a": 1, "b": [20, 30, 40]}
>>> d2 = dict(d1) # Створюємо поверхневу копію
>>> d2 ["b"][0] = "test"
>>> d1, d2 # Змінилися дві змінні
({'a': 1, 'b': ['test', 30, 40]}, {'a': 1, 'b':
['test', 30, 40]})

>>> import copy
>>> d3 = copy.deepcopy(d1)#створюємо глиб. копію
>>> d3["b"][1] = 800
>>> d1, d3 # Змінилося значення тільки в змінній
d3
({'a': 1, 'b': ['test', 30, 40]},
{'a': 1, 'b': ['test', 800, 40]})
```

Операції над словниками

1. Доступ до елементів словника здійснюється за допомогою квадратних дужок, у яких вказується ключ.
2. Як ключ можна вказати незмінюваний об'єкт: число, рядок або кортеж.

Приклад 11.

```
>>> d = {1: "int", "a": "str", (1, 2): "tuple"}
>>> d[1], d["a"], d[(1, 2)]
('int', 'str', 'tuple')
```

Якщо елемент, відповідний до зазначеного ключа, відсутній у словнику, то виконується виключення Keyerror:

Приклад 12.

```
>>> d = {"Пн": 1, "Вт": 2}; d["Ср"]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Keyerror: 'Ср'
# Доступ до неіснуючого елемента
```

Перевірка існування ключа за допомогою операторів `in` і `not in`

Оператор **`in`**. Якщо ключ знайдений, то повертається значення `True`, а якщо ні, то – `False`.

Приклад 13.

```
>>> d = { "a": 1, "b": 2}
>>> "a" in d # Ключ існує
True
>>> "c" in d # Ключ не існує
False
```

Оператор **`not in`**. Якщо ключ відсутній, повертається **`True`**, інакше – **`False`**.

Приклад 14.

```
>>> d = {"a": 1, "b": 2}
>>> "c" not in d # Ключ не існує
True
>>> "a" not in d # Ключ існує
False
```

Метод get

Формат методу:

```
get(<Ключ>[, <Значення за замовчуванням>])
```

1. За відсутності ключа повертається `None`, якщо другий параметр не заданий,

2. За відсутності ключа повертається значення другого параметра, якщо воно задане.

3. Якщо ключ присутній у словнику, то метод повертає значення, відповідне до цього ключа.

Приклад 15.

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

a=d.get("a")
print(a)
Результат: 1

a=d.get("c")
print(a)
Результат: None

a=d.get("c", 800)
print(a)
Результат: 800
```

Метод `setdefault`

Формат методу:

`setdefault(<Ключ>[,<Значення за замовчуванням>])`

1. Якщо ключ відсутній, то:

- у словнику створюється новий елемент зі значенням, зазначеним у другому параметрі;

- якщо другий параметр не зазначений, значенням нового елемента буде `None`.

2. Якщо ключ присутній у словнику, то метод повертає значення, відповідне до цього ключа.

Приклад 16.

```
d = {"a": 1, "b": 2}
a=d.setdefault("a")
print("a={0!s};  d={1!s}".format(a,d))
Результат: a=1;  d={'a': 1, 'b': 2}

a=d.setdefault("a",0)
print("a={0!s};  d={1!s}".format(a,d))
Результат: a=1;  d={'b': 2, 'a': 1}

a=d.setdefault("c")
print("a={0!s};  d={1!s}".format(a,d))
Результат:a=None;  d={'a': 1, 'c': None, 'b': 2}

a=d.setdefault("c",0)
print("a={0!s};  d={1!s}".format(a,d))
Результат: a=0;  d={'c': 0, 'b': 2, 'a': 1}
```

Модифікація словника

1. Словники є змінюваними типами даних.
2. Можна змінювати елемент по ключу.
3. Якщо елемент із зазначеним ключем відсутній у словнику, то він буде доданий у словник.

Приклад 17.

```
>>> d = {"a": 1, "b": 2}

>>> d ["a"] = 800  # Зміна елемента по ключу

>>> d["c"] = "string" # Буде доданий елемент

>>> d

{'b': 2, 'a': 800, 'c': 'string'}
```

Визначення довжини словника

Одержати кількість ключів у словнику дозволяє функція `len()`:

Приклад 18.

```
>>> d = {"a": 1, "b": 2}
>>> len(d)  #Одержуємо кількість ключів у словнику
2
```

Вилучення елемента словника

Вилучити елемент із словника можна за допомогою оператора `del`:

Приклад 19.

```
>>> d = { "a": 1, "a": 2}
>>> del d["b"]; d  # Видаляємо елемент з ключем "b"
{'a': 1}
```

Перебір елементів словника

Перебрати всі елементи словника можна за допомогою циклу `for`, хоча словники й не є послідовностями.

Два способи виводу елементів словника:

1. Перший спосіб використовує метод `keys()`, що повертає об'єкт з ключами словника.

2. У другому випадку ми просто вказуємо словник як параметр.

3. На кожній ітерації циклу буде **повертатися ключ**, за допомогою якого усередині циклу можна одержати значення, що відповідає цьому ключу.

Приклад 20.

```
#спосіб 1
d = {"x": 1, "y": 2, "z": 3}
for key in d.keys(): # Використання методу keys()
    print("{0}:{1}".format(key, d[key]), end=" ")

Результат роботи: y:2 z:3 x:1

#спосіб 2
d = {"x": 1, "y": 2, "z": 3}
for key in d: # Словники також підтримують ітерації
    print("{0!s}: {1!s},".format(key, d[key]), end=" ")

Результат роботи: z:3, x:1, y:2,
```

Сортування по ключах методом `sort()`

1. Словники є неупорядкованими структурами. Тому елементи словника виводяться в довільному порядку.

2. Щоб вивести елементи з сортуванням по ключах, слід одержати список ключів, а потім скористатися методом **`sort()`**.

Приклад 21.

```
d = {"x": 1, "y": 2, "z": 3}
k = list(d.keys())    # Одержуємо список ключів
k. sort()             # Сортуємо список ключів
for key in k:
    print("({0!s} => {1!s})".format(key, d[key]), end=" ")
```

```
Результат роботи: (x => 1) (y => 2) (z => 3)
```

Сортування по ключах функцією `sorted()`

Приклад 22.

```
d = {"x": 1, "y": 2, "z": 3}
for key in sorted(d.keys()):
    print("({0!s} => {1!s})".format(key, d[key]), end=" ")
Результат роботи:  (x => 1) (y => 2) (z => 3)
```

Тому що на кожній ітерації повертається ключ словника, функції `sorted()` можна відразу передати об'єкт словника, а не результат виконання методу `keys()` :

```
d = {"x": 1, "y": 2, "z": 3}
for key in sorted(d):
    print("({0!s} => {1!s})".format(key, d[key]), end=" ")
Результат роботи:  (x => 1) (y => 2) (z => 3)
```

Методи для роботи зі словниками (робота з ключами)

Метод `keys()` – повертає об'єкт `dict_keys`, що містить усі ключі словника. Цей об'єкт підтримує ітерації, а також операції над множинами.

Приклад 23.

```
# Одержуємо об'єкт dict values
>>> d1 = {"a": 1, "b": 2}
>>> d2 = {"a": 3, "c": 4, "d": 5}
>>> d1.keys(), d2.keys()
(dict_keys(['a', 'b']),dict_keys(['d', 'a', 'c']))

# Одержуємо кортеж списків ключів
>>> list(d1.keys()), list(d2.keys())
(['a', 'b'], ['d', 'a', 'c'])
# Перебір ключів
>>> for k in d1.keys(): print(k, end=" ")
a b
```

```
# Множина об’єднання ключів
d1, d2 = {"a": 1, "b": 2}, {"a": 3, "c": 4, "d": 5}
A = d1.keys() | d2.keys()
print(A)
```

Результат роботи: {'c', 'd', 'a', 'b'}

```
# Множина різниці ключів
A = d1.keys() - d2.keys()
print(A)
```

Результат роботи: {'b'}

```
# Множина однакових ключів
A = d1.keys() & d2.keys()
print(A)
```

Результат роботи: {'a'}

```
# Множина унікальних ключів
>>> A = d1.keys() ^ d2.keys()
{'d', 'b', 'c'}
```

Методи для роботи зі словниками (значення)

Метод **`values()`** – повертає об'єкт `dict_values`, що містить усі значення словника.
Цей об'єкт підтримує ітерації.

Приклад 24.

```
>>> d = {"a": 1, "b": 2}

>>> d.values()  # Одержуємо об'єкт dict values
dict_values([1, 2])

>>> list(d.values()) # Одержуємо список значень
[ 1, 2]

>>> [ v for v in d.values()]#Генератор значень
[ 1, 2]
```

Методи для роботи зі словниками (елементи)

Метод `items()` – повертає об'єкт `dict_items`, що містить усі ключі й значення у вигляді кортежів. Цей об'єкт підтримує ітерації.

Приклад 25.

```
>>> d = {"a": 1, "b": 2}

>>> d.items() # Одержуємо об'єкт dict items
dict_items([('a', 1), ('b', 2)])

>>> list(d.items())# Одержуємо список кортежів
[('a', 1), ('b', 2)]
```

Метод <Ключ> in <Словник>

перевіряє існування зазначеного ключа в словнику.

Якщо ключ знайдений, то повертається значення `True`, а якщо ні, то – `False`.

Приклад 26.

```
>>> d = {"a": 1, "b": 2}

>>> "a" in d # Ключ існує
True

>>> "c" in d # Ключ не існує
False
```

Метод `<Ключ> not in <Словник>`

перевіряє відсутність зазначеного ключа в словнику. Якщо такого ключа немає, то повертається значення `True`, а якщо є, то – `False`.

Приклад 27.

```
>>> d = {"a": 1,"b": 2}

>>> "c" not in d # Ключ не існує
True

>>> "a" not in d # Ключ існує
False
```

Метод `pop(<Ключ>[,<Значення за замовчуванням>])`

`pop(<Ключ>)` видаляє елемент із зазначеним ключем і повертає його значення.

`pop(<Ключ>, (Значення))` ключ відсутній, то повертається значення із другого параметра.

`pop(<Ключ>)` ключ відсутній, і другий параметр не зазначений, виконується виключення `Keyerror`.

Приклад 28.

```
d = {"a": 1, "b": 2, "c": 3}
m = d.pop("a")
print("m = {0!s};  d = {1!s}".format(m,d))
Результат:  m = 1;  d = {'b': 2, 'c': 3}

m = d.pop("a",245)
print("m = {0!s};  d = {1!s}".format(m,d))
Результат:  m = 1;  d = {'b': 2, 'c': 3}

m = d.pop("n",245)
print("m = {0!s};  d = {1!s}".format(m,d))
Результат:m = 245;d = {'c': 3, 'b': 2, 'a': 1}

m = d.pop("n")
print("m = {0!s};  d = {1!s}".format(m,d))

KeyError: 'n'
```

Метод `popitem()`

Цей метод видаляє довільний елемент і повертає кортеж із ключа й значення.

Якщо словник порожній, виконується виключення **`Keyerror`**.

Приклад 29.

```
>>> d = {"a": 1, "b": 2}

>>> d.popitem() # Видаляємо довільний елемент
('a', 1)

>>> d.popitem() # Видаляємо довільний елемент
('b', 2)

>>> d.popitem()#Словник порожній. Виконується виключення
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Keyerror: 'popitem(): dictionary is empty'
```

Метод `clear()`

Цей метод видаляє всі елементи словника.

Метод нічого не повертає як значення.

Приклад 30.

```
>>> d = {"a": 1, "b": 2}

>>> d.clear() # Видаляємо всі елементи

>>> d         # Словник тепер порожній

{ }
```

Метод `update ()` – додає елементи в словник.
Метод змінює поточний словник і нічого не повертає.

Формати методу:

```
update(<Ключ1>=<Значення1>[, ... , <Ключn>=<Значеннян>])

update(<Словник>)

update(<Список кортежів з двома елементами>)

update(<Список списків з двома елементами>)
```

Якщо елемент із зазначеним ключем уже присутній у словнику, то його значення буде перезаписано.

Приклад 31.

```
d = {"a": 1, "b": 2}
d.update(c=3, d=4, f=123)
print(d)
{'f': 123, 'c': 3, 'b': 2, 'd': 4, 'a': 1}

d.update({"c": 10, "d": 20}) # Словник
print(d)# Значення елементів перезаписані
{'d': 20, 'b': 2, 'a': 1, 'f': 123, 'c': 10}


d.update([("d", 80), ("e", 6)])#Список кортежів
print(d)
{'d': 80, 'e': 6, 'b': 2, 'a': 1, 'f': 123, 'c': 10}


d.update([["a","str"],["i","t"]])#Список списків
print(d)
{'d':80, 'e':6, 'b':2, 'i':'t', 'a':'str', 'f':123,
'c':10}
```

Генератори словників

Синтаксис генераторів словників схожий на синтаксис генераторів списків, але має дві відмінності:

1. Вираз міститься у фігурних дужках {…}, а не у квадратних.

2. Усередині виразу перед циклом **for** вказуються два значення через двокрапку, а не одне `{a:b for…}`

Значення, розташоване ліворуч від двокрапки, стає ключем.

Значення, розташоване праворуч від двокрапки, – значенням елемента.

Простий генератор словників

Приклад 33.

```
# Список з ключами
>>> keys = ["Anna", "Maria","Bonni"]

# Список зі значеннями
>>> values = [18, 21, 50]

>>> {k:v for (k, v) in zip(keys, values)}
{'Maria': 21, 'Bonni': 50, 'Anna': 18}

>>> {k:10 for k in keys}
{'Maria': 10, 'Bonni': 10, 'Anna': 10}
```

Генератор словників зі складною структурою

Генератори словників можуть мати складну структуру.

Вони можуть складатися з декількох вкладених циклів **for** і (або) містити оператор розгалуження **if** після циклу.

Створимо новий словник, що містить тільки елементи з парними значеннями, з початкового словника:

Приклад 34.

```
d={"Maxim":11, "Bart":12, "Dago":15, "Whit": 18}
p={k: v for (k,v) in d.items() if v % 2 == 0}
print(p)
```

```
Результат: {'Bart': 12, 'Whit': 18}
```