PYTHON ZBIORY

Grzegorz Danilewicz

CECHY ZBIORU (SET) W PYTHON

- Kolekcja!
- Elementy zbioru mogą być tego samego lub różnych typów
- Nieuporządkowany – kolejność elementów nie jest zachowywana, gdy wykonywane są operacje na elementach zbioru
- Elementy zbioru są unikalne (nie można umieścić dwóch takich samych elementów w zbiorze)
- Elementy zbioru nie mogą być typu pozwalającego na zmianę

KREACJA ZBIORÓW

KREACJA ZBIORU W PYTHON

1. Przez konstruktor set()

```
In[1]: x = set()
In[2]: x
Out[2]: set()
In[3]: type(x)
Out[3]: set
```

5

KREACJA ZBIORU W PYTHON

1. Przez konstruktor set()

```
In[1]: x = set()
In[2]: x
Out[2]: set()
In[3]: type(x)
Out[3]: set
```

Zbiór pusty

6

KREACJA ZBIORU W PYTHON

1. Przez konstruktor set()

```
In[1]: x = set()
In[2]: x
Out[2]: set()
In[3]: type(x)
Out[3]: set
```

Zatem - tworzenie pustego zbioru!

7

KREACJA ZBIORU W PYTHON

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: x = set([1, 2])
In[2]: x
Out[2]: {1, 2}
In[3]: type(x)
Out[3]: set
```

8

KREACJA ZBIORU W PYTHON

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: x = set([1, 2])
In[2]: x
Out[2]: {1, 2}
In[3]: type(x)
Out[3]: set
```

Lista!

Zbiór!

KREACJA ZBIORU W PYTHON

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: x = set([1])
In[2]: x
Out[2]: {1}
In[3]: type(x)
Out[3]: set
```

Lista!

Zbiór!

KREACJA ZBIORU W PYTHON

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: x = set((1, 2))
In[2]: x
Out[2]: {1, 2}
In[3]: type(x)
Out[3]: set
```


KREACJA ZBIORU W PYTHON

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: x = set((1, 2))
In[2]: x
Out[2]: {1, 2}
In[3]: type(x)
Out[3]: set
```

Krotka!

Zbiór!

KREACJA ZBIORU W PYTHON

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: x = set((1))
TypeError Traceback (most recent call last)
Input In [1], in <cell line: 1>()
----> 1 x = set((1))
TypeError: 'int' object is not iterable
```

KREACJA ZBIORU W PYTHON

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: x = set((1))
TypeError Traceback (most recent call last)
Input In [1], in <cell line: 1>()
----> 1 x = set((1))
TypeError: 'int' object is not iterable
```

Błąd!

KREACJA ZBIORU W PYTHON

Ale!!!

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: x = set((1,))
In[2]: x
Out[2]: {1}
```

Krotka!

KREACJA ZBIORU W PYTHON

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: x = set({1, 2})
In[2]: x
Out[2]: {1, 2}
In[3]: type(x)
Out[3]: set
```

Zbiór!

Zbiór!

KREACJA ZBIORU W PYTHON

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: x = set({1})
In[2]: x
Out[2]: {1}
```

Zbiór!

KREACJA ZBIORU W PYTHON

Uwaga!!!
Łańcuch znaków też jest iterowany!

2. Przez konstruktor z argumentem set(<typ iterowany>)

```
In[1]: lancuch = 'coś'
In[2]: list(lancuch)
Out[2]: ['c', 'o', 'ś']
In[3]: set(lancuch)
Out[3]: {'c', 'o', 'ś'}
```

Zbiór!
(w ogólności kolejność nie musi być zachowana)

KREACJA ZBIORU W PYTHON

3. Z definicji

```
In[1]: x = {1, 2}
In[2]: x
Out[2]: {1, 2}
```


KREACJA ZBIORU W PYTHON

3. Z definicji

```
In[1]: x = {'aaa', 'bbb', 'ccc'}
In[2]: x
Out[2]: {'aaa', 'bbb', 'ccc'}
```


KREACJA ZBIORU W PYTHON

3. Z definicji

```
In[1]: x = {'a', 'b', 'c', 'a'}
In[2]: x
Out[2]: {'a', 'b', 'c'}
```

21

UWAGA – PUŁAPKI

```
In[1]: {'abca'}
Out[1]: {'abca'}
In[2]: set('abca')
Out[2]: {'a', 'b', 'c'}
```

To nie jest to samo!!!

22

UWAGA – PUŁAPKI

```
In[1]: x = set()
In[2]: x
Out[2]: set()
In[3]: type(x)
Out[3]: set
```

Zbiór pusty

23

UWAGA – PUŁAPKI

Ale!!!

```
In[1]: x = set()
In[2]: x
Out[2]: set()
In[3]: type(x)
Out[3]: set
In[4]: x = {}
In[5]: type(x)
Out[5]: dict
```

24

UWAGA – PUŁAPKI

Ale!!!

```
In[1]: x = set()
In[2]: x
Out[2]: set()
In[3]: type(x)
Out[3]: set
In[4]: x = {}
In[5]: type(x)
Out[5]: dict
```

To nie jest zbiór pusty!!!

25

ELEMENTY ZBIORU

- zbior_A = {1, 2, 3, 4}
- zbior_B = {'słowo1', 'słowo2', 'słowo3'}

- zbior_mix_A = {1, 'słowo2', 2.71, None}
- zbior_mix_B = {1, 'słowo2', (7, 8, 9), None}

26

ELEMENTY ZBIORU

- zbior_A = {1, 2, 3, 4}
- zbior_B = {'słowo1', 'słowo2', 'słowo3'}

- zbior_mix_A = {1, 'słowo2', 2.71, None}
- zbior_mix_B = {1, 'słowo2', **(7, 8, 9)**, None}

Krotka – typ niezmienny!

27

ELEMENTY ZBIORU

- Listy i słowniki to typy zmienne – nie mogą być elementami zbioru!!!

28

DŁUGOŚĆ ZBIORU ≡ LICZBA ELEMENTÓW

- len(zbior)

```
In[1]: x = set()
In[2]: len(x)
Out[2]: 0
```

29

OPERATORY

30

OPERATOR **IN** I **NOT IN**

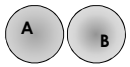
- Sprawdzenie, czy element należy (in) lub nie należy (not in) do zbioru

```
In[1]: x = {1, 2, 3, 4, 5, 6, 1}
In[2]: 1 in x
Out[2]: True
In[3]: 7 not in x
Out[3]: True
```

31

OPERACJE NA ZBIORACH W PYTHON

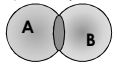
- Dwa sposoby:
1. Operatory
2. Metody klasy set



Suma zbiorów (union)

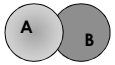
$A \cup B = \{x: x \in A \vee x \in B\}$

Część wspólna (intersection)



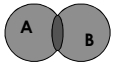
$A \cap B = \{x: x \in A \wedge x \in B\}$

Różnica zbiorów (difference)



$B \backslash A = \{x \in B: x \notin A\}$

Różnica symetryczna (symmetric difference)



$A \triangle B = (A \backslash B) \cup (B \backslash A)$

32

SUMA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_2 = {2, 4, 5}
In[3]: set_1 | set_2
Out[3]: {1, 2, 3, 4, 5}
```

33

SUMA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_2 = {2, 4, 5}
In[3]: set_1 | set_2
Out[3]: {1, 2, 3, 4, 5}
```

Uwagi:
- Operator |
- Oba argumenty muszą być zbiorami
- Wynik jest zbiorem i nie zawiera duplikatów
- Można obliczać sumę więcej niż dwóch zbiorów (a | b | c | ...)

34

SUMA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_1.union([2, 4, 5])
Out[2]: {1, 2, 3, 4, 5}
```

35

SUMA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_1.union([2, 4, 5])
Out[2]: {1, 2, 3, 4, 5}
```

Uwagi:
- Metoda union(<typ_iterowany>)
- Drugi argument może być typem iterowanym (a nie zbiorem)
- Wynik jest zbiorem i nie zawiera duplikatów
- Zbiór, dla którego wywołano metodę nie zmienia się
- Można obliczać sumę więcej niż dwóch zbiorów (a.union(b, c, ...))

36

CZĘŚĆ WSPÓLNA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_2 = {2, 4, 5}
In[3]: set_1 & set_2
Out[3]: {2}
```

CZĘŚĆ WSPÓLNA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_2 = {2, 4, 5}
In[3]: set_1 & set_2
Out[3]: {2}
```

Uwagi:
- Operator &
- Oba argumenty muszą być zbiorami
- Wynik jest zbiorem
- Można obliczać iloczyn więcej niż dwóch zbiorów (a & b & c & …)

CZĘŚĆ WSPÓLNA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_1.intersection([2, 4, 5])
Out[2]: {2}
```


CZĘŚĆ WSPÓLNA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_1.intersection([2, 4, 5])
Out[2]: {2}
```

Uwagi:
- Metoda intersection(<typ_iterowany>)
- Drugi argument może być typem iterowanym (a nie zbiorem)
- Wynik jest zbiorem
- Zbiór, dla którego wywołano metodę nie zmienia się
- Można obliczać iloczyn więcej niż dwóch zbiorów (a.intersection(b, c, …))

RÓŻNICA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_2 = {2, 4, 5}
In[3]: set_1 - set_2
Out[3]: {1, 3}
```

41

RÓŻNICA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_2 = {2, 4, 5}
In[3]: set_1 - set_2
Out[3]: {1, 3}
```

Uwagi:
- Operator –
- Oba argumenty muszą być zbiorami
- Wynik jest zbiorem
- Można obliczać różnicę więcej niż dwóch zbiorów (a – b – c – …) – obliczenia od lewej do prawej

42

RÓŻNICA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_1.difference([2, 4, 5])
Out[2]: {1, 3}
```

43

RÓŻNICA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_1.difference([2, 4, 5])
Out[2]: {1, 3}
```

Uwagi:
- Metoda difference(<typ_iterowany>)
- Drugi argument może być typem iterowanym (a nie zbiorem)
- Wynik jest zbiorem
- Zbiór, dla którego wywołano metodę nie zmienia się
- Można obliczać różnicę więcej niż dwóch zbiorów (a.difference(b, c, …))

44

SYMETRYCZNA RÓŻNICA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_2 = {2, 4, 5}
In[3]: set_1 ^ set_2
Out[3]: {1, 3, 4, 5}
```

45

SYMETRYCZNA RÓŻNICA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_2 = {2, 4, 5}
In[3]: set_1 ^ set_2
Out[3]: {1, 3, 4, 5}
```

Uwagi:
- Operator ^
- Oba argumenty muszą być zbiorami
- Wynik jest zbiorem
- Można obliczać różnicę więcej niż dwóch zbiorów (a ^ b ^ c ^ ...) – obliczenia od lewej do prawej

46

SYMETRYCZNA RÓŻNICA ZBIORÓW

```
In[1]: set_1 = {1, 2, 3}
In[2]: set_1.symmetric_difference([2, 4, 5])
Out[2]: {1, 3, 4, 5}
```

Uwagi:
- Metoda symmetric_difference(<typ_iterowany>)
- Drugi argument może być typem iterowanym (a nie zbiorem)
- Wynik jest zbiorem
- Zbiór, dla którego wywołano metodę nie zmienia się

47

ZMIENNOŚĆ ZBIORU

48

ZMIENNOŚĆ ZBIORU

- Chociaż elementami zbiorów mogą być tylko typy niezmienne, to same zbiory są zmienne
- ['__and__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__iand__', '__init__', '__init_subclass__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__xor__', '**add**', '**clear**', 'copy', 'difference', '**difference_update**', '**discard**', 'intersection', '**intersection_update**', 'isdisjoint', 'issubset', 'issuperset', '**pop**', '**remove**', 'symmetric_difference', '**symmetric_difference_update**', 'union', '**update**']

49

ZBIORY NIEZMIENNE (ZAMROŻONE)

50

ZBIORY ZAMROŻONE (FROZENSET)

- Zbiory, których zawartości nie można zmieniać
- Wykorzystują konstruktor klasy frozenset()
- Wykorzystywane wszędzie tam gdzie potrzebny jest typ niezmienny (na przykład zwykły zbiór nie może być elementem innego zbioru, ale zbiór zamrożony już tak)
- Nie działają metody związane z modyfikacją zawartości zbioru
- Ale…

51

OPERACJE ROZSZERZONE NA ZBIORACH ZAMROŻONYCH

```
In[1]: mroz = frozenset([1, 2, 3])
In[2]: set_1 = {3, 4, 5}
In[3]: mroz &= set_1
In[4]: mroz
Out[4]: frozenset({3})
```

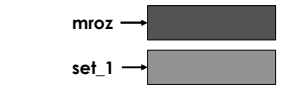
52

OPERACJE ROZSZERZONE NA ZBIORACH ZAMROŻONYCH

```
In[1]: mroz = frozenset([1, 2, 3])
In[2]: set_1 = {3, 4, 5}
In[3]: mroz &= set_1
In[4]: mroz
Out[4]: frozenset({3})
```

OPERACJE ROZSZERZONE NA ZBIORACH ZAMROŻONYCH

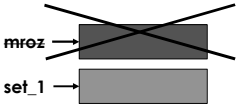
```
In[1]: mroz = frozenset([1, 2, 3])
In[2]: set_1 = {3, 4, 5}
In[3]: mroz &= set_1
In[4]: mroz
Out[4]: frozenset({3})
```



mroz &= **set_1** → **mroz** = **mroz** & **set_1**

OPERACJE ROZSZERZONE NA ZBIORACH ZAMROŻONYCH

```
In[1]: mroz = frozenset([1, 2, 3])
In[2]: set_1 = {3, 4, 5}
In[3]: mroz &= set_1
In[4]: mroz
Out[4]: frozenset({3})
```



mroz &= **set_1** → **mroz** = **mroz** & **set_1**