

Programowanie proceduralne

INP001210WL

rok akademicki 2020/21

semestr letni

Wykład 5

Karol Tarnowski

karol.tarnowski@pwr.edu.pl

L-1 p. 220



Plan prezentacji

- Słowniki
 - tworzenie słownika
 - operacje na słowniku - wybrane operatory, funkcje, metody
- Zbiory
 - tworzenie zbiorów
 - operacje na zbiorach - wybrane operatory, funkcje, metody



Słownik

- Słownik to obiekt przechowujący dane w postaci par: klucz-wartość.



Utworzenie słownika

- Przykład polecenia tworzącego słownik:

```
dictionary = {1: 'pn', 2: 'wt', 3: 'śr' }
```

- Słownik można wyświetlić funkcją `print()`

```
print(dictionary)
```

```
>>> dictionary = {1: 'pn', 2: 'wt', 3: 'śr' }  
>>> print(dictionary)  
{1: 'pn', 2: 'wt', 3: 'śr' }  
>>>
```



Różne typy wartości

- Wartości przechowywane w słowniku mogą być różnych

```
>>> movie = {'title':'The Shawshank Redemption', 'year':1994, 'grade':8.8}
>>> print(movie)
{'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.8}
>>>
```



Różne typy kluczy

- Klucze również mogą być różnych typów

```
>>> divisors = {1:[1], 'two':[1,2], '6':[1,2,3,6]}
>>> print(divisors)
{1: [1], 'two': [1, 2], '6': [1, 2, 3, 6]}
>>>
```

Pobieranie wartości ze słownika

- Elementy słownika nie są przechowywane w ustalonej kolejności
- Nie ma możliwości dostępu do wartości na podstawie pozycji
- Do pobrania wartości używa się klucza

dictionary[key]

```
>>> dictionary[2]
'wt'
>>> movie['title']
'The Shawshank Redemption'
>>> divisors['6']
[1, 2, 3, 6]
>>>
```

Pobieranie wartości ze słownika

- Użycie złego klucza powoduje błąd **KeyError**

```
>>> dictionary[4]
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    dictionary[4]
KeyError: 4
>>> divisors['six']
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    divisors['six']
KeyError: 'six'
>>>
```


Pobieranie wartości ze słownika

- Wielkość liter ma znaczenie

```
>>> movie['Title']
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    movie['Title']
KeyError: 'Title'
>>>
```

Sprawdzanie wartości w słowniku

- Aby uniknąć błędu można sprawdzić, czy klucz istnieje w słowniku
- Do sprawdzenia można wykorzystać operator **in**

```
>>> if 'Title' in movie:  
    print(movie['Title'])
```

```
>>> if 'title' in movie:  
    print(movie['title'])
```

```
The Shawshank Redemption
```

```
>>>
```

Sprawdzanie wartości w słowniku

- Można również użyć operatora `not in`

```
>>> if 3 not in divisors:  
    print('3 nie jest kluczem.')
```

```
3 nie jest kluczem.
```

```
>>>
```



Dodanie elementu do słownika

- Słownik jest obiektem modyfikowalnym
- Polecenie

dictionary[key] = value

- zmodyfikuje wartość dla klucza **key** jeśli ten klucz istnieje w słowniku
- doda nową parę do słownika w przeciwnym przypadku

```
>>> print(movie)
{'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.8}
>>> movie['grade'] = 8.7
>>> print(movie)
{'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.7}
>>> movie['country'] = 'United States'
>>> print(movie)
{'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.7, 'country': 'United States'}
>>>
```



Usunięcie elementu ze słownika

- Polecenie `del` pozwala usunąć parę klucz-wartość **`del dictionary[key]`**

```
>>> print(movie)
{'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.7, 'country': 'United States'}
>>> del movie['country']
>>> print(movie)
{'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.7}
>>>
```

Usunięcie elementu ze słownika

- Jeśli klucz nie istnieje, to pojawi się błąd **KeyError**

```
>>> del movie['garde']
Traceback (most recent call last):
  File "<pyshell#44>", line 1, in <module>
    del movie['garde']
KeyError: 'garde'
>>>
```



Sprawdzenie liczby elementów

- Do sprawdzenia liczby elementów można wykorzystać funkcję `len()`

```
>>> print(movie)
{'title': 'The Shawshank Redemption', 'year': 1994, 'grade': 8.7}
>>> len(movie)
3
>>>
```

Klucze muszą być niemodyfikowalne

- Obiekt modyfikowalny nie może być kluczem

```
>>> mixed = {} #utworzenie nowego słownika
>>> mixed['abc'] = 1 #dodanie pary klucz wartość
>>> #klucz jest napisem, wartość liczbą
>>> mixed[4] = 'def' #klucz jest liczbą, wartość napisem
>>> mixed[(2,3,5,7)] = [2,3,5,7] #klucz jest krotką, wartość lista
>>> print(mixed)
{'abc': 1, 4: 'def', (2, 3, 5, 7): [2, 3, 5, 7]}
>>> mixed[[11,13]] = 'to się nie uda' #klucz nie może być lista
Traceback (most recent call last):
  File "<pyshell#63>", line 1, in <module>
    mixed[[11,13]] = 'to się nie uda' #klucz nie może być lista
TypeError: unhashable type: 'list'
>>>
```


Tworzenie słownika funkcją `dict()`

- Funkcja `dict()` tworzy słownik
- Funkcja `dict()` pozwala tworzyć słownik przez argumenty nazwane - nazwa argumentu staje się kluczem, a wartość argumentu staje się wartością w parze
- Funkcja `dict()` pozwala tworzyć słownik z obiektu, po którym można iterować (np. z listy)



Tworzenie słownika funkcją `dict()`

```
>>> a = dict(one=1, two=2, three=3)
>>> print(a)
{'one': 1, 'two': 2, 'three': 3}
>>> b = {'one':1, 'two':2, 'three':3 }
>>> print(b)
{'one': 1, 'two': 2, 'three': 3}
>>> c = dict([('two',2), ('one',1), ('three',3)])
>>> print(c)
{'two': 2, 'one': 1, 'three': 3}
>>> d = dict({'three':3, 'one':1, 'two':2})
>>> print(d)
{'three': 3, 'one': 1, 'two': 2}
>>> a == b == c == d
True
>>>
```



Słowniki i pętla for

- Wykorzystując pętle `for` można prowadzić iterację przez wszystkie klucze słownika

```
>>> for key in movie:  
        print(key)
```

```
title  
year  
grade  
>>>
```



Słowniki i pętla for

- Dla każdego klucza można sprawdzić wartość,

```
>>> for key in movie:  
        print(key, movie[key])
```

```
title The Shawshank Redemption  
year 1994  
grade 8.7  
>>>
```

nie jest to dobra metoda...



Słowniki i pętla for

- Ten sam efekt można osiągnąć wykorzystując metodę `items()`, która zwraca widok słownika

```
>>> for key, value in movie.items():  
        print(key, value)
```

```
title The Shawshank Redemption  
year 1994  
grade 8.7  
>>>
```



Metoda `items()`

- Metoda `items()` zwraca widok słownika (sekwencję zawierającą dwuelementowe krotki przechowujące pary klucz-wartość)

```
>>> print(movie.items())  
dict_items([('title', 'The Shawshank Redemption'), ('year', 1994), ('grade', 8.7)])  
>>>
```



Metody `keys ()` i `values ()`

- Metoda `keys ()` zwraca widok słownika zawierający wszystkie klucze słownika
- Metoda `values ()` zwraca widok słownika wszystkie wartości słownika

```
>>> print(movie.keys())
dict_keys(['title', 'year', 'grade'])
>>> print(movie.values())
dict_values(['The Shawshank Redemption', 1994, 8.7])
>>>
```



Metoda `get()`

- Metoda `get()` pozwala odczytać wartość ze słownika
- Jeśli klucza nie ma w słowniku zwraca wartość domyślną

`dictionary.get(key, default)`

```
>>> print(movie.get('title', 'unknown key'))
The Shawshank Redemption
>>> print(movie.get('country', 'unknown key'))
unknown key
>>>
```




Metoda pop ()

- Metoda `pop ()` działa podobnie do metody `get ()`
- Dodatkowo usuwa parę klucz-wartość ze słownika

```
>>> print(movie.pop('country', 'unknown key'))
unknown key
>>> print(movie.pop('title', 'unknown key'))
The Shawshank Redemption
>>> print(movie)
{'year': 1994, 'grade': 8.7}
>>>
```



Metoda `popitem()`

- Metoda `popitem()` działa podobnie do metody `pop()`
- Zwraca jedną parę klucz-wartość (LIFO) i usuwa wpis ze słownika

```
>>> k, v = movie.popitem()
>>> print(k, v)
grade 8.7
>>> print(movie)
{'year': 1994}
>>>
```



Metoda `clear()`

- Metoda `clear()` usuwa zawartość słownika

```
>>> movie.clear()
>>> print(movie)
{}
>>>
```



Zbiór

- Zbiór to obiekt przechowujący kolekcję unikatowych wartości



Funkcja `set()`

- Do utworzenia zbioru służy funkcja `set()`
- Funkcja `set()` może być wywołana bez argumentów
- Funkcja `set()` może być wywołana z argumentem w postaci listy
- Funkcja `set()` może być wywołana z argumentem w postaci łańcucha znakowego



Funkcja set ()

```
>>> empty_set = set()
>>> set_abc = set(['a', 'b', 'c'])
>>> char_set = set('abc')
>>>
>>> print(empty_set)
set()
>>> print(set_abc)
{'a', 'b', 'c'}
>>> print(char_set)
{'a', 'b', 'c'}
>>>
```



Funkcja set ()

- Zbiór zawiera tylko unikatowe elementy

```
>>> set('WPPT PWr')  
{'r', 'W', 'P', 'T', ' '}  
>>>
```



Funkcja `set()`

- Funkcja `set()` pobiera maksymalnie 1 argument

```
>>> set('WPPT', 'PW_r')
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#28>", line 1, in <module>
```

```
    set('WPPT', 'PW_r')
```

```
TypeError: set expected at most 1 arguments, got 2
```

```
>>>
```

- Przykład tworzenia zbioru zawierającego łańcuchy znakowe

```
>>> set(['WPPT', 'PW_r'])
```

```
{'WPPT', 'PW_r'}
```

```
>>>
```




Sprawdzenie liczby elementów

- Do sprawdzenia liczby elementów można wykorzystać funkcję `len()`

```
>>> chars = set('WPPT PWr')  
>>> len(chars)  
5
```

Dodawanie elementów do zbioru

- Dodawanie elementów metodą `add()`

```
>>> myset = set()
>>> myset.add(1)
>>> myset.add(2)
>>> myset.add(3)
>>> myset
{1, 2, 3}
>>> myset.add(2)
>>> myset
{1, 2, 3}
>>>
```

Dodawanie elementów do zbioru

- Dodawanie elementów metodą `update()`

```
>>> myset.update([4, 5, 6])  
>>> myset  
{1, 2, 3, 4, 5, 6}  
>>>
```

Dodawanie elementów do zbioru

- Dodawanie elementów metodą `update()`

```
>>> myset2 = set([7, 8, 9])
>>> myset.update(myset2)
>>> myset
{1, 2, 3, 4, 5, 6, 7, 8, 9}
>>>
```

Usuwanie elementów do zbioru

- Usuwanie elementów metodą `remove()`

```
>>> myset.remove(2)
>>> myset
{1, 3, 4, 5, 6, 7, 8, 9}
>>> myset.remove(2)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    myset.remove(2)
KeyError: 2
>>>
```

Usuwanie elementów do zbioru

- Usuwanie elementów metodą `discard()`

```
>>> myset.discard(3)
>>> myset
{1, 4, 5, 6, 7, 8, 9}
>>> myset.discard(3)
>>>
```

- Metoda `discard()` nie zgłasza błędu, gdy elementu nie ma w zbiorze

Usuwanie wszystkich elementów zbioru

- Czyszczenie zbioru metodą `clear()`

```
>>> myset.clear()  
>>> myset  
set()  
>>>
```



Przechodzenie przez wszystkie elementy zbioru

- Do przejścia po elementach zbioru możemy wykorzystać pętlę `for`

```
>>> myset = set([-2, -1, 0, 1, 2])  
>>> for el in myset:  
    print(el)
```

```
0  
1  
2  
-1  
-2  
>>>
```


Sprawdzanie należenia do zbioru

- Na zbiorach działają operatory **in** oraz **not in**

```
>>> myset
{0, 1, 2, -1, -2}
>>> 1 in myset
True
>>> -5 in myset
False
>>> 1 not in myset
False
>>> -5 not in myset
True
>>>
```



Łączenie zbiorów

- Do łączenia zbiorów można wykorzystać metodę `union()` lub operator `|`

```
>>> set1 = {1, 2, 3, 4}
>>> set2 = {3, 4, 5, 6}
>>> set3 = set1.union(set2)
>>> print(set3)
{1, 2, 3, 4, 5, 6}
>>> set4 = set1 | set2
>>> print(set4)
{1, 2, 3, 4, 5, 6}
>>>
```



Część wspólna zbiorów

- Do wyznaczenia części wspólnej można wykorzystać metodę `intersection()` lub operator `&`

```
>>> set5 = set1.intersection(set2)
```

```
>>> set5
```

```
{3, 4}
```

```
>>> set6 = set1 & set2
```

```
>>> set6
```

```
{3, 4}
```

```
>>>
```



Różnica zbiorów

- Do wyznaczenia różnicy zbiorów można wykorzystać metodę `difference()` lub operator `-`

```
>>> set7 = set1.difference(set2)
>>> set7
{1, 2}
>>> set8 = set1 - set2
>>> set8
{1, 2}
>>>
```



Różnica symetryczna zbiorów

- Do wyznaczenia różnicy symetrycznej zbiorów można wykorzystać metodę **`symmetric_difference()`** lub operator **`^`**

```
>>> set9 = set1.symmetric_difference(set2)
```

```
>>> set9
```

```
{1, 2, 5, 6}
```

```
>>> set10 = set1 ^ set2
```

```
>>> set10
```

```
{1, 2, 5, 6}
```

```
>>>
```



Wyszukiwanie podzbioru

- Do sprawdzenia zawierania się zbiorów można wykorzystać metody `issubset()` oraz `issuperset()` lub operatory `<=` oraz `>=`

```
>>> set1 = set('abcdef')
>>> set2 = set('ae')
>>> set2.issubset(set1)
True
>>> set1.issuperset(set2)
True
>>> set2<=set1
True
>>> set1>=set2
True
>>>
```



Podsumowanie

- Słowniki
 - tworzenie słownika
 - operacje na słowniku - wybrane operatory, funkcje, metody
- Zbiory
 - tworzenie zbiorów
 - operacje na zbiorach - wybrane operatory, funkcje, metody