Wykład5

December 11, 2019

1 Zbiory (set)

[ ]:
```
help(set)
```

[2]:
```
lista_1=[]
zbior_1=set()
```

[3]:
```
print(type(lista_1),type(zbior_1))
```

```
<class 'list'> <class 'set'>
```

[4]:
```
lista_1.append("abc")
zbior_1.add("abc")
```

[5]:
```
print(lista_1)
print(zbior_1)
```

```
['abc']
{'abc'}
```

[8]:
```
lista_1.append("abc")
zbior_1.add("abc")
```

[9]:
```
print(lista_1)
print(zbior_1)
```

```
['abc', 'abc', 'abc']
{'abc'}
```

[10]:
```
lista_1.append("abc","cvb")
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)
```

```
    <ipython-input-10-264109e39785> in <module>
----> 1 lista_1.append("abc","cvb")


TypeError: append() takes exactly one argument (2 given)
```

[11]:
```
zbior_1.add("abc","cvb")
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

    <ipython-input-11-87244561543d> in <module>
----> 1 zbior_1.add("abc","cvb")


TypeError: add() takes exactly one argument (2 given)
```

[12]:
```
lista_1.append(["abc","cvb"])
```

[13]:
```
lista_1
```

[13]:
```
['abc', 'abc', 'abc', ['abc', 'cvb']]
```

[14]:
```
zbior_1.add(["abc","cvb"])
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

    <ipython-input-14-43ccb4c348c6> in <module>
----> 1 zbior_1.add(["abc","cvb"])


TypeError: unhashable type: 'list'
```

[15]:
```
zbior_1.add(("abc","cvb"))
```

[16]:
```
zbior_1
```

[16]:
```
{('abc', 'cvb'), 'abc'}
```

[17]:
```
zbior_1.discard("abc")
```

[18]:
```
zbior_1
```

[18]:
```
{('abc', 'cvb')}
```

[19]:
```
zbior_1[0]
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-19-bd1da17996e1> in <module>
----> 1 zbior_1[0]


TypeError: 'set' object does not support indexing
```

[20]:
```
zbior_1={"abc"}
```

[21]:
```
"abc" in lista_1
```

[21]:
```
True
```

[22]:
```
"abc" in zbior_1
```

[22]:
```
True
```

[23]:
```
lista_2=list("cvb")
zbior_2=set("cvb")
```

[24]:
```
print(lista_2)
print(zbior_2)
```

```
['c', 'v', 'b']
{'v', 'c', 'b'}
```

[25]:
```
lista_3=["a","b","c","d"]
zbior_3={"a","b","c","d"}
```

[26]:
```
type(zbior_3)
```

[26]:
```
set
```

1.0.1 Operacje na zbiorach

[27]:
```
len(zbior_2)
```

[27]:
```
3
```

[28]:
```
zbior_2 | zbior_3  #suma
```

[28]:
```
{'a', 'b', 'c', 'd', 'v'}
```

[29]:
```
zbior_2 - zbior_3  #różnica
```

[29]:
```
{'v'}
```

[30]:
```
zbior_2 & zbior_3 #część wspólna
```

[30]:
```
{'b', 'c'}
```

[31]:
```
zbior_2.union(lista_3)
```

[31]:
```
{'a', 'b', 'c', 'd', 'v'}
```

[32]:
```
zbior_2.difference(lista_3)
```

[32]:
```
{'v'}
```

[33]:
```
zbior_2.intersection(lista_3)
```

[33]:
```
{'b', 'c'}
```

[34]:
```
zbior_3
```

[34]:
```
{'a', 'b', 'c', 'd'}
```

[35]:
```
zbior_3 > {"b","c"} #Nadzbiór
```

[35]:
```
True
```

[36]:
```
zbior_1
```

[36]:
```
{'abc'}
```

[37]:
```
zbior_2
```

[37]:
```
{'b', 'c', 'v'}
```

[38]:
```
zbior_1|=zbior_2
```

[39]:
```
zbior_1
```

[39]:
```
{'abc', 'b', 'c', 'v'}
```

1.0.2 Zastosowanie: Usuwanie duplikatów

[40]:
```
L_1=[1,2,1,3,2,4,5]
print(set(L_1))
```

```
{1, 2, 3, 4, 5}
```

[41]:
```
L_1=list(set(L_1))
print(L_1)
```

```
[1, 2, 3, 4, 5]
```

1.0.3 Zbiory "zamrożone"

[42]:
```
F_zbior=frozenset("cvb")
```

[43]:
```
F_zbior
```

[43]:
```
frozenset({'b', 'c', 'v'})
```

[44]:
```
F_zbior.add("abc")
```

```
␣
↪---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-44-5371799e0df0> in <module>
----> 1 F_zbior.add("abc")


AttributeError: 'frozenset' object has no attribute 'add'
```

2 Słowniki (dict)

[ ]:
```
help(dict)
```

[45]:
```
slownik_1={}
slownik_1["policja"]=997
slownik_1["straz"]=998
slownik_1["pogotowie"]=999
```

[46]:
```
slownik_1
```

[46]:
```
{'pogotowie': 999, 'policja': 997, 'straz': 998}
```

W przeciwienstwie do list, gdzie przypisania poza granicami listy jest zakazane, przypisania do nowych kluczy słownika tworzą te klucze.

[47]:
```
slownik_1["policja"]
```

[47]:
```
997
```

[48]:
```
len(slownik_1)
```

[48]:
```
3
```

[49]:
```
list(slownik_1.keys())
```

[49]:
```
['straz', 'pogotowie', 'policja']
```

[50]:
```
list(slownik_1.values())
```

[50]:
```
[998, 999, 997]
```

[51]:
```
"straz" in slownik_1
```

[51]:
```
True
```

[52]:
```
slownik_1["dane_osobowe"]={"imie":"Jan","nazwisko":"Kowalski"}
```

[53]:
```
slownik_1
```

[53]:
```
{'dane_osobowe': {'imie': 'Jan', 'nazwisko': 'Kowalski'},
 'pogotowie': 999,
 'policja': 997,
 'straz': 998}
```

[54]:
```
slownik_1["wyksztalcenie"]=["magister", "inzynier"]
```

[55]:
```
slownik_1
```

[55]:
```
{'dane_osobowe': {'imie': 'Jan', 'nazwisko': 'Kowalski'},
 'pogotowie': 999,
```

```
 'policja': 997,
 'straz': 998,
 'wyksztalcenie': ['magister', 'inzynier']}
```

[56]:
```
slownik_1.items()
```

[56]:
```
dict_items([('dane_osobowe', {'imie': 'Jan', 'nazwisko': 'Kowalski'}), ('straz',
998), ('pogotowie', 999), ('wyksztalcenie', ['magister', 'inzynier']),
('policja', 997)])
```

[57]:
```
slownik_1["klucz"]
```

```
␣
↪---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-57-7acd3941f77d> in <module>
----> 1 slownik_1["klucz"]


KeyError: 'klucz'
```

[58]:
```
print(slownik_1.get("klucz"))
```

```
None
```

[59]:
```
slownik_1.get("klucz","brak klucza")
```

[59]:
```
'brak klucza'
```

[60]:
```
slownik_1.get("wyksztalcenie","brak klucza")
```

[60]:
```
['magister', 'inzynier']
```

2.0.1 Pakowanie - zip

[61]:
```
list(zip(['a', 'c', 'b'], [1, 2, 3]))
```

[61]:
```
[('a', 1), ('c', 2), ('b', 3)]
```

[62]:
```
D = dict(zip(['a', 'c', 'b'], [1, 2, 3]))
```

[63]:
```
D
```

[63]:
```
{'a': 1, 'b': 3, 'c': 2}
```

2.0.2 Co jeszcze jest możliwe ...

[64]:
```
dir(slownik_1)
```

[64]:
```
['__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__init__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'clear',
 'copy',
 'fromkeys',
 'get',
 'items',
 'keys',
 'pop',
 'popitem',
 'setdefault',
 'update',
 'values']
```

[65]:
```
slownik_2={}
slownik_2.fromkeys([1,2,3,4,5,6,7],0)
```

[65]:
```
{1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0}
```

[ ]:
```
vars()
```

2.0.3 Przykład zastosowania

[68]:
```
rzym = { 1000:"M", 900:"CM", 500:"D", 400:"CD", 100:"C", 90:"XC", 50:"L",
40:"XL", 10:"X", 9:"IX", 5:"V", 4:"IV", 1:"I" }
```

[69]:
```
def liczby_rzymckie():
    x = int(input("Podaj liczbę całkowitą:"))
    print("Liczba",str(x),"w notacji rzymskiej to:",end=" ")
    r = list(rzym.keys())
    r.sort()
    r.reverse()
    lr = ""
    for i in r:
        while i <= x:
            lr += rzym[i]
            x -= i
    print(lr)
```

[70]:
```
liczby_rzymckie()
```

```
Podaj liczbę całkowitą:33
Liczba 33 w notacji rzymskiej to: XXXIII
```

[71]:
```
liczby_rzymckie()
```

```
Podaj liczbę całkowitą:999
Liczba 999 w notacji rzymskiej to: CMXCIX
```

[72]:
```
liczby_rzymckie()
```

```
Podaj liczbę całkowitą:2019
Liczba 2019 w notacji rzymskiej to: MMXIX
```

[ ]:
```

```

[73]:
```
def funkcja_1(*args,**kwargs):
    print(args)
    print(kwargs)
```

[74]:
```
funkcja_1(1,2,3,"abc",a=1,d=3,g="g")
```

```
(1, 2, 3, 'abc')
{'d': 3, 'a': 1, 'g': 'g'}
```

[75]:
```
D
```

[75]:
```
{'a': 1, 'b': 3, 'c': 2}
```

[76]:
```
funkcja_1(1,2,3,D)
```

```
(1, 2, 3, {'a': 1, 'c': 2, 'b': 3})
{}
```

[77]:
```
funkcja_1(1,2,3,**D) #rozpakowanie słownika
```

```
(1, 2, 3)
{'a': 1, 'c': 2, 'b': 3}
```

3 Formatowanie Tekstu

[78]:
```
help(print)
```

```
Help on built-in function print in module builtins:

print(…)
    print(value, …, sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

[79]:
```
a,b,c=1.5,2,3
print(a,b,c,sep="...",end="~koniec")
```

```
1.5…2…3~koniec
```

[80]:
```
print("a =",a,"b =",b,"c=",c)
```

```
a = 1.5 b = 2 c= 3
```

[ ]:
```
print("a = "+str(a)+" b = "+str(b)+" c= "+str(c))
```

[81]:
```
print("a = {} b = {} c= {}".format(a,b,c))
```

```
a = 1.5 b = 2 c= 3
```

[82]:
```
print("b = {1} c = {2} a= {0}".format(a,b,c))
```

```
b = 2 c = 3 a= 1.5
```

[83]:
```
print("a = {:f} b = {:f} c= {:d}".format(a,b,c))
```

```
a = 1.500000 b = 2.000000 c= 3
```

[84]:
```
d=1234567890
print("a = {:f} b = {:f} c= {:e}".format(a,b,d))
```

```
a = 1.500000 b = 2.000000 c= 1.234568e+09
```

[85]:
```
d=1234567890
print("a = {:f} b = {:f} c= {:e}".format(a,d,b))
```

```
a = 1.500000 b = 1234567890.000000 c= 2.000000e+00
```

[86]:
```
d=1234567890
print("a = {a:f} b = {b:d} c= {d:e}".format(a=2.5,d=9876543210,b=2))
```

```
a = 2.500000 b = 2 c= 9.876543e+09
```

[87]:
```
from math import pi
print("pi = {}".format(pi))
```

```
pi = 3.141592653589793
```

[88]:
```
print("pi = {:.2f}".format(pi))
print("pi = {:.2f}".format(pi))
```

```
pi = 3.14
pi = 3.14
```

[89]:
```
print("pi = {:.50f}".format(pi))
```

```
pi = 3.14159265358979311599796346854418516159057617187500
```

[90]:
```
print("{:*^4}".format("Start"))
print("{:*^6}".format("Start"))
print("{:*^10}".format("Start"))
print("{:*^15}".format("Start"))
```

```
Start
Start*
```

```
**Start***
*****Start*****
```

[91]:
```
print("{:*<10}".format("Start"))
print("{:*>10}".format("Start"))
```

```
Start*****
*****Start
```

[92]:
```
for x in range(1,11):
    print("{0:2d} {1:4d} {2:6d}".format(x,x**2,x**3))
```

```
 1    1      1
 2    4      8
 3    9     27
 4   16     64
 5   25    125
 6   36    216
 7   49    343
 8   64    512
 9   81    729
10  100   1000
```

[ ]: