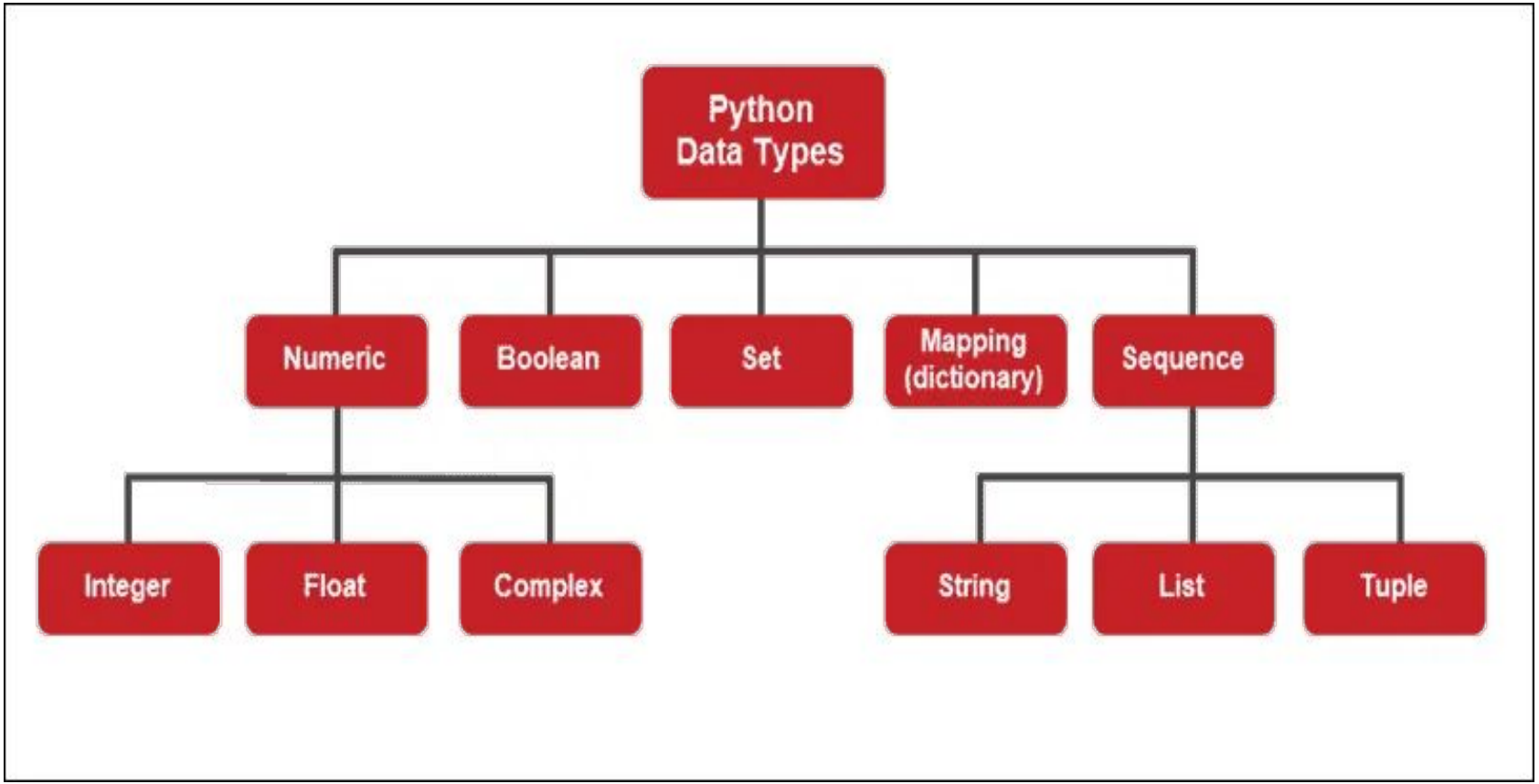
Python Programming

Lecture 2 - data types

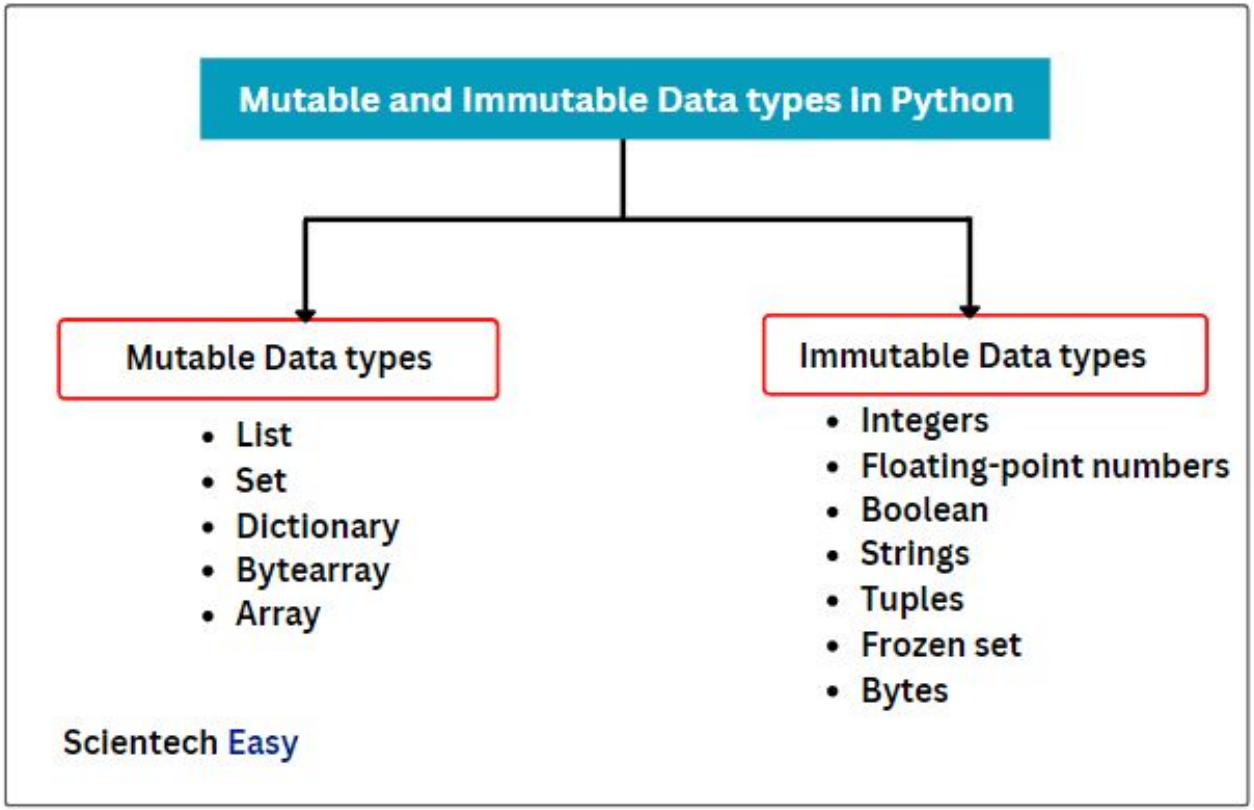
PhD Aneta Polewko-Klim
Wyższa Szkoła Ekonomiczna w Białymstoku

Standard data types



https://medium.com/

Data Structures: mutable and immutable



Mutable object is an object whose value (state) can be modified after it is defined.

Immutable object is an object whose value (state) can not be modified after it is defined; assigning a new value to a variable creates a new object in memory

Numeric data type

- **integer (*int*)**
- **float (*float*) - real numeric**
- **complex number (*complex*)**

Note: variables are typed dynamically

```
var_integer = 20 # assigning value to variable
var_float = 20.0  #  the equal sign (=) is used to assign a value to a variable
var_complex = 2+15j


print(type(var_float))  # type() function either returns the type of the object
<class 'float'>

# Assign multiple values to multiple variables

x1, x2 = 20, 30
```

Basic operations: numeric data

```
1 + 2 # result is integer
1 + 2.0 # result is float
2 - 3
-2 * 3
7 / 2 # result is float

# type conversion of variable in Python

# the process of converting a data type into another data type
int(3.5) # convert float to int      float(2) # convert int to float
2.0
complex(1,1) or 1+1j  # complex number
1+1i
(1+2j).real # real part              (1+2j).imag # image part
1.0                                    2.0
```

Basic operations: numeric data

```
abs(-10) # absolute
abs(1+1j)
(1+1j).conjugate() # conjugate (1-1j)
9 // 2 # divide without remainder
4

5 % 2 # modulo operator operation is used to get the remainder of a division
divmod(10, 3) # pair of numbers (10 // 3 i 10 % 3)
(3, 1)

pow(2, 3)  lub   2 ** 3    # power operator 2^3
8
pow(2, 3.0) # power operator 2^3
8.0
```

Math package

```
import math

# ceil returns the smallest integer greater than or equal to the input
real number
math.ceil(1.8)
2

dir(math) # list of functions of math module

 'acos',  'acosh',  'asin',  'asinh',  'atan',  'atan2',  'atanh',
'ceil', 'copysign',  'cos',  'cosh', 'degrees',  'e',  'erf',  'erfc',
'exp', 'expm1',  'fabs',  'factorial',  'floor',  'fmod',  'frexp',
'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite',
'isinf','isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2',
'modf', 'nan','pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc'
```

Python Cheat Sheet

https://pythononeliners.com

Boolean

The Boolean data type is a truth value, either `True` or `False`.

The Boolean operators ordered by priority:
`not x` → "if x is False, then x, else y"
`x and y` → "if x is False, then x, else y"
`x or y` → "if x is False, then y, else x"

These comparison operators evaluate to `True`:
`1 < 2 and 0 <= 1 and 3 > 2 and 2 >=2 and 1 == 1 and 1 != 0 # True`

```
## 1. Boolean Operations
x, y = True, False
print(x and not y) # True
print(not x and y or x) # True

## 2. If condition evaluates to False
if None or 0 or 0.0 or '' or [] or {} or set():
    # None, 0, 0.0, empty strings, or empty
    # container types are evaluated to False
    print("Dead code") # Not reached
```

Integer, Float

An integer is a positive or negative number without floating point (e.g. `3`). A float is a positive or negative number with floating point precision (e.g. `3.14159265359`).

The '`//`' operator performs integer division. The result is an integer value that is rounded towards the smaller integer number (e.g. `3 // 2 == 1`).

```
## 3. Arithmetic Operations
x, y = 3, 2
print(x + y) # = 5
print(x - y) # = 1
print(x * y) # = 6
print(x / y) # = 1.5
print(x // y) # = 1
print(x % y) # = 1s
print(-x) # = -3
print(abs(-x)) # = 3
print(int(3.9)) # = 3
print(float(3)) # = 3.0
print(x ** y) # = 9
```

Sequential type of data: string

- string is data surrounded by inverted comas ‘ ‘ or “ “
- **string(*str*)** `var_string1 = ‘10’ var_string2 = “10” var_string3 = ‘John’`
  - **Zapamiętaj:** Indeksowanie w Pythonie rozpoczyna się od 0, Python rozróżnia małe i duże litery
  - Zmienna **string** przechowuje ciąg znaków, w którym każdy znak ma określoną pozycję.

```
# citations

print('Student: "Life is Beautiful" ')

print("Student: \"Life is Beautiful\" ") # escape character is a backslash \

>> Student: "Life is Beautiful"
```

Basic operations: string

Remember: Python uses zero-based indexing and distinguishes between lowercase and uppercase letters

```
word = 'INFORMATIC'
print(word[0]) # each char in string has a specific position.
>> I


print(word[0:3])
>> INF


print(word[0:10:3]) # sequence[min:max:step]
>> IOAC


print(word[1:])
>> NFORMATIC
```

Basic operations: string

```
other example
word[2::4] # from 3 item to end, step 4
word[::2] # from first item to end, step 2

word = 'INFORMATIC'

# możemy odwoływać się do elementów od końca
print(word[-1])     # last character of a string
>> C

print(word[-2:]) # two last character of a string
>> IC

print(word[:-2]) # omit two last character of a string
>> INFORMAT         # wszystkie pomijając dwa ostatnie
```

Basic operations: string

- `\n` - new line
- `\r` - return of the cursor
- `\t` - tabulate
- `\v` - vertical tabulate

```
print("Line 1\n Line 2")    # new line
>> Line 1
   Line 2
```

```
print("Line1 \t Line1.")
>> Line1    Line2   #   Tu wstawiamy tabulator     widzisz różnicę.
```

Long text:

```
text = "Python is programming language \       # use backslash
      high level \
     general purpose"
print(text)
>> Python is programming language high level general purpose
```

Python String Methods

Method	Description
capitalize()	Converts the first character to upper case
count()	Returns the number of times a specified value occurs in a string
endswith()	Returns true if the string ends with the specified value
find()	Searches the string for a specified value and returns the position of where it was found
replace()	Returns a string where a specified value is replaced with a specified value
split()	Splits the string at the specified separator, and returns a list
startswith()	Returns true if the string starts with the specified value

More string methods see: link

Python String Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string

Python Cheat Sheet

https://pythononeliners.com

String

Python Strings are sequences of characters.

The four main ways to create strings are the following.

1. Single quotes

```
'Yes'
```

2. Double quotes

```
"Yes"
```

3. Triple quotes (multi-line)

```
"""Yes
We Can"""
```

4. String method

```
str(5) == '5' # True
```

5. Concatenation

```
"Ma" + "hatma" # 'Mahatma'
```

These are whitespace characters in strings.

- Newline \n
- Space \s
- Tab \t

```
## 4. Indexing and Slicing
s = "The youngest pope was 11 years old"
print(s[0])       # 'T'
print(s[1:3])     # 'he'
print(s[-3:-1])   # 'ol'
print(s[-3:])     # 'old'
x = s.split()     # creates string array of words
print(x[-3] + " " + x[-1] + " " + x[2] + "s")
                  # '11 old popes'

## 5. Most Important String Methods
y = "    This is lazy\t\n   "
print(y.strip()) # Remove Whitespace: 'This is lazy'
print("DrDre".lower()) # Lowercase: 'drdre'
print("attention".upper()) # Uppercase: 'ATTENTION'
print("smartphone".startswith("smart")) # True
print("smartphone".endswith("phone")) # True
print("another".find("other")) # Match index: 2
print("cheat".replace("ch", "m")) # 'meat'
print(','.join(["F", "B", "I"])) # 'F,B,I'
print(len("Rumpelstiltskin")) # String length: 15
print("ear" in "earth") # Contains: True
```

Sequential type of data: list

- has a dynamic size
- list contents is *mutable*

We can create lists by using 3 methods:

- list is a collection of data surrounded by [ ] brackets i.e.

```
list1 = ['python','c++','java']
```

- use **list()** constructor

```
list2 = list(['python','c++','java'])
```

- lista składana (*list comprehension*)

```
word_list = ["dom", "kwiat", "szkoła"] # definiujemy listę
number_letter = [len(w) for w in word_list]
print(number_letter)
```

Sekwencyjne typy danych: listy

- Lista może zawierać zmienne dowolnego typu
- posiada dynamiczny rozmiar
- zawartość listy jest "zmienna" (obiekt *mutable*)

Tworzymy na 3 sposoby:

- nawiasy kwadratowe `lista = ['python','c++','java']`
- korzystając z konstruktora klasy **list**
  `list_program = list(['python','c++','java'])`
- lista składana (*list comprehension*)

```
word_list = ["dom", "kwiat", "szkoła"] # definiujemy listę
number_letter = [len(w) for w in word_list]
print(number_letter)
```

Sequential type of data: tuple

- tuples are used to store multiple items in a single variable.
- tuple is a built-in data type in Python used to store collections of data.
- tuple is a collection that is ordered and unchangeable.

Creating a tuple (3 methods):

- use **( )** brackets `tuple_1 = ('python','c++','java')`
- use **tuple()** constructor `tuple_1 = tuple(['python','c++','java'])`
- ciąg elementów oddzielonych przecinkiem

Note: use tuple if the number of object in sequence is constant in your program

Sekwencyjne typy danych: krotka

- Krotka może zawierać zmienne dowolnego typu
- posiada stały rozmiar

Tworzymy na 3 sposoby:

- nawiasy okrągłe `krotka = ('python','c++','java')`
- korzystając z konstruktora klasy **tuple**
  `tuple_program = tuple(['python','c++','java'])`
- ciąg elementów oddzielonych przecinkiem
- zawartość krotki jest "niezmienna" (obiekt *immutable*)
- jeśli w programie Twoja sekwencja obiektów jest stała używaj krotek (są)

Range

- **range()** is immutable sequence of numbers
- takes less memory than a list or a tuple
  (stores only following information: start, stop, and step

range(start, stop) # default step equals 1
range(start, stop, step)

```
vector numbers = range(5) # from 0 to 5
print(vector numbers)
x = range(0, 5)

# We need to dump range objects into a list if we want to see the
result

print(list(x))
```

Basic operations on sequence data

- *in* - prawda jeśli element należy do sekwencji, inaczej fałsz

```
a = 'informatics'
"o" in a
```

```
True
```

- *not in* - fałsz jeśli element należy do sekwencji, inaczej prawda

```
"m" not in a
```

```
False
```

Basic operations on sequence data

- *splot* - combining multiple string variables

```
'Like'+'Python'
'LikePython'
```

```
[4,5,6]+[1,2,3]
[4, 5, 6, 1, 2, 3]
```

- *multiplication by a constant* (operacja nie obowiązuje dla *range*)

```
2*'Python'
'PythonPython'
```

Basic operations on sequence data

- *indexing*

```
list1 = [10, 20, 30, 40, "informatics"]
list1[:2] # all elements before the element with index 2
>> [10, 20]


list1[4][-2:] # two last chars of the element with index 4
>> 'ka'


print(list(range(3, 15, 3)[:2])) # two first chars of the element with >>
[3, 6]

max("frog") # minimal element    min([10, 2, 15])  # maximal element
>> 'r'                                  >> 2

len([10, 3, 15])  # length
>> 3
```

Basic operations on sequence data

- *zmiana/dodawanie elementów*

```
lista = [10, 20, 30, 40, "informatics"]
lista[4] = 'Python' # zmiana
[10, 20, 30, 40, 'Python']
lista[2:3]= [0,0,0]  # zmiana elementu
[10, 20, 30, 0, 0, 0, 40, 'Python']
```

- *usuwanie elementów*

```
del lista[1:2]
[10, 0, 0, 40, 'informatyka']
```

Wybrane użyteczne metody dla sekwencji typu *mutable*

- dodawanie elementów do listy

```
lista1 = [1, 2, 3, 4]
lista1.append(7) # metoda append dodaje element na koniec listy
[1, 2, 3, 4, 7]
lista1.append([17, 18, 19]) # lista dodana jako element listy
[1, 2, 3, 4, 7, [17, 18, 19]]
lista2 = [1, 2, 3]
lista2.extend([10, 20, 30]) # extend rozwija listę
[1, 2, 3, 10, 20, 30]
```

- usuwanie elementów do listy

```
lista.clear() # usuwa wszystkie elementy
[]
```

Wybrane użyteczne metody dla sekwencji typu *mutable*

copy elements

```
list1 = [1, 2, 3, 4]
list2 = list1.copy()
```

item substitution

```
list1.insert(2, 50) # insert 50 to 3 position of list
[1, 2, 50, 3, 4]
```

remove element of list

```
list1.pop(1) # remove 2 element of list, show this element
list1.remove(2) # remove value 2 (first appearance in the list)
```

Wybrane użyteczne metody dla sekwencji typu *mutable*

- proste sortowanie elementów sekwencji

```
lista = ['d', 'A', 'a', 'b', 'C']
lista.sort()  # sortowanie alfabetyczne lub rosnąco
['A', 'C', 'a', 'b', 'd']
```

- sortowanie wg funkcji

```
lista.sort(reverse=False,key=str.lower) # sortuj wg funkcji key
[ 'a', 'b', 'd','A', 'C']
```

```
def myFun(e):  # tworzymy własną funkcję
return len(e)

cars = ['Ford', 'Mitsubishi', 'BMW', 'VW']
cars.sort(reverse=True, key=myFunc)
lista.sort(reverse=False, key=str.lower) # sortuj wg funkcji key
(ustawi)
[ 'a', 'b', 'd','A', 'C']
```

Dictionary

- dictionary is is *immutable object,* can contain different type variables
- dictionary maps hashable objects into arbitrary objects, i.e.: key: value
  key - constant *key value* - arbitrary object

key could be tuple (not list)

Hashowanie jest procesem konwersji dużej ilości danych na znacznie mniejszą ilość (zazwyczaj pojedynczą liczbę całkowitą) w sposób powtarzalny, skraca to czas wyszukiwania danych.

Creating a tuple (3 methods):

- use **{ }** brackets **`{key : value}`**

```
dict1 = {'IDLE': 'PyCharm', 'jezyk': 'Python'}
```

- use **dict()** constructor **`dict(key : value)`**

```
dict1 = dict('IDLE'= 'PyCharm', 'jezyk'='Python'}
```

- use **dict()** constructor and **zip() function** **`dict([(key1, value1), (key2, value2)...])`**

```
dict1 = dict(zip(["one", "two", "three"], [1, 2, 3]))
```

Mapujące typy danych: słowniki

- Słownik może zawierać zmienne dowolnego typu
- mapuje obiekty hashowalne (*hashable*) w dowolne obiekty, czyli: *klucz: wartość*
  klucz - stały hash *wartość* - dowolny obiekt
- kluczem może być krotka, ale lista nie
- zawartość słownika jest "niezmienna" (obiekt *immutable*)

Tworzymy na 2 sposoby:

- wpisujemy w nawiasy klamrowe **`{klucz : wartość klucza}`**
  `słownik = {'IDLE': 'PyCharm', 'jezyk': 'Python'}`
- korzystając z konstruktora klasy **dict** **`dict(klucz : wartość klucza)`**
  `słownik = dict('IDLE'= 'PyCharm', 'jezyk'='Python'}`

Można również wykorzystać konstruktor klasy **dict** i funkcję **zip**

```
dict([(key1, value1), (key2, value2)...])
słownik = dict(zip(["jeden", "dwa", "trzy"], [1, 2, 3]))
```

Python Dictionary Methods

How to get value using key in dictionary Python?

- **dictionary_name[key_name]**

```
dict1 = {"John": 1980, "Grazyna": 1985, "Angelica": 2016} # data urodzin
dict1["John"]
1980
```

- **dictionary_name.get(key_name)**

```
dict1.get("Grazyna")
1980
```

A digression about the default value **None**:

```
print(dict1.get("Peter"))
None
if dict1.get("Peter"):
    print(dict1.get("Peter"))
else:
    print("No name")
```

```
dict1.keys() # display list of keys
dict1.values() # display list of values
dict1.items() # display list of key-value pairs
```

Python Dictionary Methods

- *Modification of key value*

```
dict1 = {"John": 1980, "Grazyna": 1985, "Angelica": 2016}
dict1["Angelica"] = 2018   # dict1[key]= new value
print(dict1)
{"John": 1980, "Grazyna": 1985, "Angelica": 2018}
```

- *Remove key*

```
del dict1["Angelica"]   # del dict1[key]
print(dict1)   --->   {"John": 1980, "Grazyna": 1985}
```

- *Clear dictionary*

```
dict1.clear() # wyczyść słownik
print(dict1)   --->   {}
```

Python Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Dictionary: displaying content in a loop

```
dict1 = {"John": 1980, "Grazyna": 1985, "Angelica": 2016}

    for key in dict1.keys(): #for-loop iterates over the keys
    print(key, end=' ')

    John Grazyna Angelica

    for val in dict1.values(): #loop iterates over the values
    print(val, end=' ')

    1980 1985 2016

    for key, val in dict1.items(): #loop iterates over the (key, value)
    print(key, val)

    John 1980

    Grazyna 1985

    Angelica 2016
```

Dictionary: update of dictionary in a loop

```
    dict1 = {"John": 1980, "Grazyna": 1985, "Angelica": 2016}

while True:
    person = input("Name: ")
    if not person: break # break if no input data
    year = input("Enter the year of your birth: ")
    dict1.update({person: year})    # dictionary update
```

Nested Dictionary

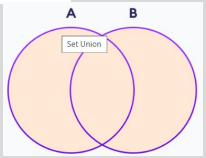
```
# value of key could be any object, such as a dictionary or list
wFabian = {"Age": 20, "Student": {'U1':"UwB",'U2':"Medical University"}}
wNikola = {"Age": 22, "Student": ["UwB", "Medical University"]}
students = {"Fabian": wFabian, "Nikola": wNikola}
print(students['Fabian'])
print(students['Fabian']['Student']['U1'])
print(students['Nikola'])
print(students['Nikola']['Student'][0])
```

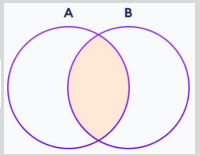
```
{'Age': 20, 'Student': {'U1': 'UwB', 'U2': 'Medical University'}}
University in Bialystok
{'Age': 22, 'Student': ['University in Bialystok', 'Medical University']}
University in Bialystok
```

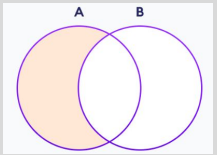
Set

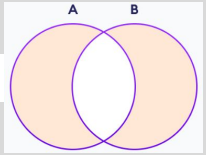
A set is an unsorted collection, without repetition, like a set in mathematics. You can perform exactly the same operations on sets as in mathematics, e.g.: sum, difference, product and symmetric difference

```
set1 = {10,2,30,4,50}   # structured programming
set2 = set()    # object-oriented programming, create the set_2 object
set2.add(1)     # calling a method on an object
set2.add(2)
print(set1 | set2)     or    set1.union(set2)
{1, 2, 50, 4, 10, 30}
print(set1 & set2)       or       set1.intersection(set2)
{2}
print(set1 - set2) or  set1.difference(set2)
{10, 4, 50, 30}
print(set1 ^ set2)  or  set1.symmetric_difference(set2)
{10,30,4,50}
```









| - vertical bar & - ampersand ^ - caret

Zbiory danych - operacje

Operacja	Wynik
len(*s*)	liczność zbioru *s*
x in *s*	sprawdzenie przynależności obiektu *x* do zbioru *s*
x not in *s*	sprawdzenie braku przynależności obiektu *x* do zbioru *s*
s.issubset(*t*)	sprawdzenie, czy każdy z elementów zbioru *s* należy do zbioru *t*;
	odpowiednikiem tej operacji jest porównanie *s* <= *t*
s.issuperset(*t*)	sprawdzenie, czy każdy z elementów zbioru *t* należy do zbioru *s*;
	odpowiednikiem tej operacji jest porównanie *s* >= *t*
s | *t*	nowy zbiór, zawierający elementy ze zbiorów *s* oraz *t*
s.union(*t*)	nowy zbiór, zawierający elementy ze zbiorów *s* oraz *t*
s & *t*	nowy zbiór, zawierający elementy wspólne dla zbiorów *s* i *t*
s.intersection(*t*)	nowy zbiór, zawierający elementy wspólne dla zbiorów *s* i *t*
s - *t*	nowy zbiór, zawierający elementy zbioru *s* z wykluczeniem elementów ze zbioru *t*
s.difference(*t*)	nowy zbiór, zawierający elementy zbioru *s* z wykluczeniem elementów ze zbioru *t*
s ^ *t*	nowy zbiór, zawierający elementy przynależące do dokładnie jednego ze zbiorów *s* lub *t*
s.symmetric_difference(*t*)	nowy zbiór, zawierający elementy przynależące do dokładnie jednego ze zbiorów *s* lub *t*
s.copy()	nowy zbiór, będący płytką kopią zbioru *s*

https://pl.python.org

Zbiory danych - operacje

Operacja	Wynik
s |= *t*	zwraca zbiór *s*, uzupełniony o elementy ze zbioru *t*
s.union_update(*t*)	zwraca zbiór *s*, uzupełniony o elementy ze zbioru *t*
s &= *t*	zwraca zbiór *s* po usunięciu z niego elementów, które nie występują w zbiorze *t*
s.intersection_update(*t*)	zwraca zbiór *s* po usunięciu z niego elementów, które nie występują w zbiorze *t*
s -= *t*	zwraca zbiór *s* po usunięciu z niego elementów, które występują w zbiorze *t*
s.difference_update(*t*)	zwraca zbiór *s* po usunięciu z niego elementów, które występują w zbiorze *t*
s ^= *t*	zwraca zbiór *s* po wypełnieniu go elementami przynależącymi do
	dokładnie jednego ze zbiorów *s* lub *t*
s.symmetric_difference_update(*t*)	zwraca zbiór *s* po wypełnieniu go elementami przynależącymi do
	dokładnie jednego ze zbiorów *s* lub *t*
s.add(*x*)	dodaje element *x* do zbioru *s*
s.remove(*x*)	usuwa element *x* ze zbioru *s*
s.discard(*x*)	usuwa element *x* ze zbioru *s*, jeśli w nim występuje
s.pop()	usuwa i zwraca wybrany dowolnie element ze zbioru *s*
s.update(*t*)	dodaje do zbioru *s* elementy ze zbioru *t*
s.clear()	usuwa ze zbioru *s* wszystkie elementy

Built-in Functions with Set

Method	Description
add()	Adds an element to the set
clear()	Removes all elements from the set
copy()	Returns a copy of the set
difference()	Returns the difference of two or more sets as a new set
difference_update()	Removes all elements of another set from this set
discard()	Removes an element from the set if it is a member. (Do nothing if the element is not in set)
intersection()	Returns the intersection of two sets as a new set
intersection_update()	Updates the set with the intersection of itself and another
isdisjoint()	Returns True if two sets have a null intersection

Python Set Methods

Method	Description
issubset()	Returns True if another set contains this set
issuperset()	Returns True if this set contains another set
pop()	Removes and returns an arbitrary set element. Raises KeyError if the set is empty
remove()	Removes an element from the set. If the element is not a member, raises a KeyError
symmetric_difference()	Returns the symmetric difference of two sets as a new set
symmetric_difference_update()	Updates a set with the symmetric difference of itself and another
union() e.g. A.union(B))	Returns the union of sets in a new set
update()	Updates the set with the union of itself and others

Python Set Methods

Function	Description
all()	Returns `True` if all elements of the set are true (or if the set is empty).
any()	Returns `True` if any element of the set is true. If the set is empty, returns `False`.
enumerate()	Returns an enumerate object. It contains the index and value for all the items of the set as a pair.
len()	Returns the length (the number of items) in the set.
max()	Returns the largest item in the set.
min()	Returns the smallest item in the set.
sorted()	Returns a new sorted list from elements in the set(does not sort the set itself).
sum()	Returns the sum of all elements in the set.