**String**

# immutable: int, float, bool, str, tuple, frozenset

# mutable types: list, set, dict

str = "Hello World!"

print(len(str)) # 12 - len - zwraca liczbę znaków (numerowane od 1)

print(type(str)) # <class 'str'>

print(str[0]) # H - wyciągnięcie znaku z ciągu znaków (numerowanie od 0. Zero to pierwszy znak)

print(str[0:5]) # Hello - wyciągnięcie znaków od 1 do 4 , 5 oznacza na który znaku ma zakończyć i jej nie pokaże

print(str[len(str) - 1])# ! - taki zapis umożliwia podanie ostatniego znaku z ciągu znaków

str2 = str + " and hello again!" # Hello World! and hello again!

print(str2[6:])# World! and hello again! - pokaże z ciągu znaków znaki od 6 do ostatniego

print(str2[::3])# HlWl deogn - pokaże co 3 literę z ciągu znaków z uwzględnieniem 1 litery

multiLine2 = "Pierwsza linia\nDruga linia\nTrzecia \t \"linia\"\\" # \ backslash - znak specjalny tzw ucieczki

# np \n - następna linia    \t - tabulator  \" - dodatkowy cudzysłów w łańcuchu znaków"

# \\ - dodatkowy backslash w łańcuchu znaków

string = "Hello World!"

print("ala".capitalize()) # Ala - zamienia pierwszą literę łańcucha znaków na wielką

print("ola ma kota, ola ma psa".count("ola")) # 2 - sprawdza ilość wystąpienia słowa w łańcuchu znaków

print(" Hello ".center(112,"\*")) # centruje łańcuch znaków(112 - ilość znaków w nowym łańcuchu, \* -dodatkowe znaki)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Hello \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

print("ola ma kota, ola ma psa".title()) # Ola Ma Kota, Ola Ma Psa - każdy ciąg po białym znaku z wielkiej litery

print( string.startswith("Hello")) # True - czy łańcuch znaków string zaczyna sie łańcuchem Hello

print( string.endswith("World!")) # True - czy łańcuch znaków string kończy sie łańcuchem World!

print(string.find("l")) # 2 -  szuka l w łańcuchu string i zwraca indeks wystąpienia

print(string.find("Ola")) # -1 -  szuka wyraz Ola w łańcuchu znaków string, jeśli nie ma zwraca wynik: -1

print(string.find("World")) # 6 - zwraca jego początek w łańcuchu znaków ("W" jest 6 licząc od 0)

pozycjaLitery = string.find("d")

print("Pozycja liter d: {}".format(pozycjaLitery)) # Pozycja liter d: 10

print("10 litera w naszym stringu to: {}".format(string[10])) # 10 litera w naszym stringu to: d

print("ola ma psa, ola ma kota".find("ola")) # 0 – zwraca indeks początku szukanego stringa

print("ola ma psa, ola ma kota".rfind("ola")) # 12 - rfind - rozpoczyna szukanie od prawej strony łańcucha znaków,

                                              # zwraca początek zaczynając od 0 od lewej strony łańcucha

print( "23456789".isalnum()) # True - sprawdza, czy łańcuch znaków składa sie z samych liter i cyfr

print( "23456789.5".isalnum()) # False,

print( "23456789 ".isalnum()) # False

print( "23456789 k".isalnum()) # False

print( "23456789kKK".isalnum()) # True

print("kotek".isalpha()) # True - sprawdza, czy łańcuch znaków składa sie z samych liter

print("123kotek".isalpha()) # False

print(" kotek".isalpha()) # False

print("test".islower()) # True - sprawdza, czy łańcuch znaków składa sie z samych małych liter

print("tesT".islower()) # False

print("123test".islower()) # True

print("TEST".isupper()) # True - sprawdza, czy łańcuch znaków składa sie z samych wielkich liter

print("TESt".isupper()) # False

print("123TEST".isupper()) # True

print("test".isspace()) # False- sprawdza, czy łańcuch znaków zawiera same białe znaki(spacja, tabulator, nowa linia)

print("   \n\n\t   ".isspace()) # True

print("-|-".join(["ala", "ola", "adam", "ania"])) # ala-|-ola-|-adam-|-ania - łączy wartości w liście z wcześniej

 # join - lącz                                                 # ustalonym przez nas łańcuchem znaków (np. -|-)

print("Hello World!".lower()) # hello world! - wszystkie litry w łańcuchu znaków zamienia na małe

print("Hello World!".upper()) # HELLO WORLD! - wszystkie litry w łańcuchu znaków zamienia na WIELKIE

print("Hello World!".swapcase()) # hELLO wORLD! - zamienia liter małe na wielkie/ wielkie na małe w łańcuchu znaków

print("   \n \t Hello    World! \n \t   ".strip()) # usuwa wszystkie białe znaki z łańcucha znaków

                                                   # (bez białych znaków między tekstem)

print("   \n \t Hello World! \n \t   ".lstrip()) # usuwa wszystkie białe znaki po lewej stronie stringa

print("   \n \t Hello World! \n \t   ".rstrip()) # usuwa wszystkie białe znaki po prawej stronie stringa

print("ola ma kota, ola ma psa".replace("ola", "Adam")) # Adam ma kota, Adam ma psa - zamienia „ola” na „Adam”

# przykłady użycia funkcji format służącej do formatowania tekstu

print("My name is {myName}, my postal code {code}, I'm from {country}".format(myName = "Michał", code = 26600, country = "Poland"))

print("My name is {0}, my postal code {1}, I'm from {2}".format("Michał", 26600, "Poland"))

print("My name is {}, my postal code {}, I'm from {}".format("Michał", 26600, "Poland"))

# kolejne przykłady użycia funkcji format służącej do formatowania tekstu (pobieramy argumenty ze zmiennej)

argument = ["Michał", 38]

tekst = "My name is {0}, I've {1} years old. {0}{1}".format(argument[0], argument[1])

tekst = "My name is {imie}, I've {wiek} years old. {imie}{wiek}".format(imie = argument[0], wiek = argument[1])

moje\_imie = "Michał"

moj\_wiek = 38

print(f"My name is {moje\_imie}, I've {moj\_wiek}. {argument}")

print(f"My name is {moje\_imie}, I've {moj\_wiek}. {argument[0]}{argument[1]}")

print(f"moved {path} to {destination}") # moved ass to work

path, destination, subtotal = "ass", "work", 0.0

print("rachunek całkowity: $%.2f" % subtotal) # rachunek całkowity: $0.00 - %.2f % subtotal - wyświetli liczbę 2 miejsca po przecinku

**List**

lista = [] # pusta lista

lista = [1, 2, 3, "c", "d", "e"] # lista może zawierać różne typy argumentów

print(type(lista)) # <class 'list'>

lista[3] = 4 # zmiana 4 elementu listy z indeksem[3]("c") na nowy element 4

print(lista + ["f", 'g']) # [1, 2, 3, 4, 'd', 'e', 'f', 'g'] - można dodać inną listę do już istniejącej

print(lista \* 2) # [1, 2, 3, 4, 'd', 'e', 1, 2, 3, 4, 'd', 'e']- zwielokrotnienie listy 2 razy

print("ilość elementów: ", len(lista))# ilość elementów:  6

lista = [1, 2, 3, 'c', 'd', 'e']

print(lista[1:5]) # [2, 3, 4, 'd'] - wycinek listy(krotki też)od indeksu 1 do 4 (kończący 5 nie jest wyświetlany)

lista = [1,2,4,8,16,32,64,128,256,512,1024]

print(lista[-6:-2]) # [32, 64, 128, 256] - możemy pobrać wycinek listy od końca. Indeksowanie wtedy zaczynamy

# od -1( w tym przypadku 1024). Podobnie jak w normalnym wycinku drugi indeks(-2 czyli 512) nie jest wyświetlany

print(lista[6:])# [64, 128, 256, 512, 1024] - pokaże wycinek listy od 6 do ostatniego

print(lista[::3])# [1, 8, 64, 512] - pokaże co 3 wartość z listy z uwzględnieniem 1 indeksu

print(lista[2:11:2])# [4, 16, 64, 256, 1024] - pokaże co 2 wartość z wycinka z uwzględnieniem 1 indeksu

print(lista[::-1])# [1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1] - odwróci kolejność argumentów w liście

lista = [1, 2, 3, 'c', 'd', 'e']

lista.append("f") # append - metoda dodaje do listy argument (na samym końcu)

print(lista) # [1, 2, 3, 4, 'd', 'e', 'f']

lista.append(["g", "h"]) # dołączenie do listy innej listy

print(lista) # [1, 2, 3, 4, 'd', 'e', 'f', ['g', 'h']]

lista\_1 = [-1,0,1,2,3,4,5,6,7,8]

print(lista\_1[2:-1]) # [1, 2, 3, 4, 5, 6, 7]

print(lista\_1.pop())# [-1, 0, 1, 2, 3, 4, 5, 6, 7] - usuwa z listy ostatni argument

lista\_1[1:3] = "A", "B" # zamienimy indeks 1 i 2 (czyli drugą{0} i trzecią cyfrę{1}) na litery A i B

print(lista\_1) # [-1, 'A', 'B', 2, 3, 4, 5, 6, 7]

lista\_1[3:5] = "A", "B", "C" # jeśli ilośc deklarowanych do zmiany argumentów jest wieksza(3), niż to wynika z zakresu

                             #([3:5] - 2 arg), to te 2 zostaną nadpisane, a nadmiarowy argument zostanie dodany do listy

print(lista\_1) # [-1, 'A', 'B', 'A', 'B', 'C', 4, 5, 6, 7]

lista\_1[4:] = [] # usuniemy z listy od 4 argumentu do końca (a właściwie dodamy pustą listę)

print(lista\_1) # [-1, 'A', 'B', 'A']

lista\_1 += lista # do lista\_1 dodamy lista

print(lista\_1) # [-1, 'A', 'B', 'A', 1, 2, 3, 4, 'd', 'e', 'f', ['g', 'h']]

print(lista) # [1, 2, 3, 4, 'd', 'e', 'f', ['g', 'h']]

print(lista[7]) # ['g', 'h']

print(lista[7][1]) # h

lista.insert(4, 3) # insert - metoda dodająca w tym przypadku cyfrę 3 do listy nadając jej indeks 4

print(lista) # [1, 2, 3, 4, 3, 'd', 'e', 'f', ['g', 'h']]

print("ilość:", lista.count(3)) # ilość: 2 - count(liczyć, zliczać)- poda nam ilość wystąpienia w liście cyfry 3

print("index:", lista.index("f")) # index: 7 - podaje indeks wybranego argumentu listy( w tym przypadku "f")

lista.remove("e") # remove - usuwa argument z listy ( w tym przypadku "e")

print(lista) # [1, 2, 3, 4, 3, 'd', 'f', ['g', 'h']]

lista2 = [1,4,6,-2,5,0,-8]

lista3 = ["ala", "rafał", "adam", "zenon", "grzegorz", "filip", "aga"]

# wartości minimalne/maksymalne z listy

print("Min:", min(lista2)) # Min: -8

print("Max:", max(lista2)) # Max: 6

print("Min:", min(lista3)) # Min: adam

print("Max:", max(lista3)) # Max: zenon

lista2.sort() # sortowanie listy od najmniejszej do największej

print(lista2) # [-8, -2, 0, 1, 4, 5, 6]

lista3.sort()

print(lista3) # ['adam', 'aga', 'ala', 'filip', 'grzegorz', 'rafał', 'zenon']

lista2.reverse() # odwrócenie kolejności listy ( przepisanie od końca po koleji do początku)

print(lista2) # [6, 5, 4, 1, 0, -2, -8]

lista3.reverse()

print(lista3) # ['zenon', 'rafał', 'grzegorz', 'filip', 'ala', 'aga', 'adam']

lista2.clear() # czyszczenie listy

print(lista2) # []

# metoda split służy do utworzenia listy z ciągu znaków ( wprowadzonych z klawiatury za pomocą polecenia input)

# z pominięciem separatorów występujących w tym ciągu

value1 = input("podaj cyfry, stringi po przecinku: ") # 1,2,345,ola,w2

value2 = input("podaj cyfry, stringi po separatorze | : ") # 1|67|ghj

list1 = value1.split(",")🡪 print(list1) # ['1', '2', '345', 'ola', 'w2']

list2 = value2.split("|")🡪 print(list2) # ['1', '67', 'ghj']

# metoda wyciągnięcia z listy argumentów listy

color\_list = ["Red","Green","White" ,"Black"]

a = "%s %s"%(color\_list[0],color\_list[-1])# Red Black

color\_list = ["Red","Green","White" ,"Black"]

print( "%s %s %s %s"%(color\_list[0],color\_list[1],color\_list[2],color\_list[3])) # Red Green White Black

# operacje na listach

lista = list(range(10)) # tworzy listę [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(lista)

# metody tworzenia list dzięki pętli for i instrukcji if

nowaLista = [i \* 2 for i in lista] # [0, 2, 4, 6, 8, 10, 12, 14, 16, 18] - wykonuje iterację każdego elementu

# listy i wykonuje dla niego operację ( w tym przypadku i\*2)

nowaLista2 = [i + 2 for i in lista if i % 2 == 0] # [2, 4, 6, 8, 10]

nowaLista3 = [i + 1 for i in lista if i % 2 == 0] # [1, 3, 5, 7, 9]

# przypadki powyżej: iteracja wykonywana jest dopiero wtedy, gdy zostanie spełniony warunek if. Co ciekawe,

# jak pokazuje nowaLista3, kolejność odczytywania operacji jest taka, że najpierw sprawdzany jest warunek

# if ( i%2 == 0 co oznacza reszta z dzielenia przez 2 rowna 0) i tylko te wartości są iterowane, które

# go spełnią ( w tym przypadku 0,2,4,6,8)

# metoda all sprawdza czy założony warunek (i % 2 == 0) jest spelniony dla wsystkich argumentów listy.

# Jeśli tak, zwraca True, jeśli nie, zwraca False. Poniżej zastosowano instrukcję if, wiec zwróci nam odpowiedni łańcuch znaków

lista = [11,20,25,30,41]

if all([i % 2 == 0 for i in lista]):

    print("Wszystkie parzyste")

else:

    print("Niewszystkie parzyste")

# podobnie jak wyżej, tylko any szuka chociaż jednego argumentu, który spełnia założony warunek

if any([i % 2 == 0 for i in lista]):

    print("Chociaż jedna parzysta")

else:

    print("Wszystkie nieparzyste")

# enumerate numeruje nam od 0 każdy argument z listy. Wynikiem jest krotka składająca się numeru przypisanego

# przez enumerate i argumentu listy

for i in enumerate(lista):

print(i)

(0, 11)

(1, 20)

(2, 25)

(3, 30)

(4, 41)

# poniżej dokonaliśmy sformatowania wyniku na bardziej przejżysty i zaczynający się od 1 (a nie 0)

for i in enumerate(lista):

print(i[0] + 1, "-", i[1])

1 - 11

2 - 20

3 - 25

4 - 30

5 - 41

# metoda difference (różnica) dla zbiorów, lista też jest zbiorem.

# Aby zadziałało na liście, trzeba ją przerobić na zbiór (set)

color\_list\_1 = set(["White", "Black", "Red"])

color\_list\_2 = set(["Red", "Green"])

print(color\_list\_1.difference(color\_list\_2)) # {'Black', 'White'} zwraca różnice dla zbiorów

print(color\_list\_2.difference(color\_list\_1)) # {'Green'}

**Tuple**

data = ("Ala", "Ola", "Kasia")# krotka(inaczej tablica), jest niemutowalna, nie można jej zmieniać ani edytować

names = data + ("Rafał",) # ('Ala', 'Ola', 'Kasia', 'Rafał')

print(type(names)) # <class 'tuple'>

numbers = 1,2,3 # krotkę można utworzyć bez nawiasów, jeśli ma ona posiadać kilka elementów

emptyTuple = () # pusta krotka

print(names[1]) # Ola

print(names[-1]) # Rafał- wyświetla od końca krotki

print(names[1:3]) # ('Ola', 'Kasia')

cars = (("Dodge","Ford"), ("Pontiac",)) # krotka złożona z 2 krotek

print(cars[0][0]) # wyświetli Dodge

print(cars[0][1]) # wyświetli Ford

print(cars[1][0]) # wyświetli Pontiac

del cars # kasowanie krotki, odniesienie się do nie po skasowaniu wywoła błąd

# del names[0] - wyświetli błąd, kod nie zadziała. Nie można skasować elementu krotki, KROTKA JEST NIEMUTOWALNA

tuple1 = (0,1,2,3,4) + (5,) + tuple([6,7]) # (0, 1, 2, 3, 4, 5, 6, 7) twozrenie nowej krotki z 2 krotek i listy

print((1,2) \* 4) # (1, 2, 1, 2, 1, 2, 1, 2) zwielokrotnienie krotki

print( 9 in tuple1 ) # False - czy 9 jest w krotce tuple1

print(tuple1[2]) # 2 - zwraca 3 element krotki ( elementy numerowane są od 0)

print(len(tuple1)) # 8 - ile elementów jest w krotce

print(min(tuple1)) # 0 - nalmniejszy element krotki

print(max(tuple1)) # 7 - największy element krotki

print("elementów: ", tuple1.count(6)) # elementów:  1 - zlicza ilość wystąpień wartości 6 w krotce

print("index: ", tuple1.index(6)) # index:  6 -pokazuję indeks wystąpienia wartości 6 w krotce indeksując od 0

# metoda wyciągnięcia z krotki danych

exam\_st\_date = (11,12,2014)

print(type(exam\_st\_date))

print( "The examination will start from : %i / %i / %i"%exam\_st\_date)# The examination will start from : 11 / 12 / 2014

tuple1 = (0, 1, 2, 3, 4, 5, 6, 7)

print(sum(tuple1)) # 28, suma poszczególnych argumentów krotki

a, b, c, d, e, f, g, h = tuple1 # tworzenie uproszczone wielu zmiennych zawartych w krotce

print("a:{}, b:{}, c:{}, d:{}, e:{}, f:{}, g:{}, h:{}".format(a,b,c,d,e,f,g,h)) # a:0, b:1, c:2, d:3, e:4, f:5, g:6, h:7

print("a:%s, b:%s, c:%s, d:%s, e:%s, f:%s, g:%s, h:%s"%(tuple1)) # a:0, b:1, c:2, d:3, e:4, f:5, g:6, h:7

# drugi sposób jest kompatybilny ze starszym Pythonem 2

# unboxing (rozpakowywanie) krotki- można stosować też dla innych kolekcji(np list)

a, b = (2, 5) 🡪 print(a) #2 , print(b) #5

x = 10 , y = 20

x, y = y, x

print("x: ", x) # x: 20

print("y: ", y) # y: 10

start, \*wszystko, koniec = (1,2,3,4,5,6,7,8,9) # \* działa tak, że zmienna \*wszystko może wziąć nieskończenie

# wiele argumentów i zostaną one wyświetlone w [liście]. Jeśli dodamy jeszcze inne zmienne j.w , to one pobiorą

# sobie odpowiednie argumenty zdodnie z ich indeksem

print(start) # 1 print(wszystko) # [2, 3, 4, 5, 6, 7, 8] print(koniec)# 9

**Dictionary (słownik)**

contacts = {                    # słownik od dictionary <class 'dict'>

    "Ola" : "ola@exaple.com",

    "Daniel" : 30,

    "Ania" : "ania@example.com"

}

contacts["Rafał"] = "rafał@example.com" # dodanie kolejnego rekordu do słownika

print(contacts["Ola"])       # ola@exaple.com - wywołanie ze słownika rekordu Ola

print(type(contacts))        # wywołanie typy <class 'dict'>

print(len(contacts))         # 4 - ilość rekordów w słowniku (klucz + wrtość)

print( contacts.keys())   # dict\_keys(['Ola', 'Daniel', 'Ania', 'Rafał']) wyświetla wszystkie klucze ze słownika

print( contacts.values()) # dict\_values(['ola@exaple.com', 30, 'ania@example.com', 'rafał@example.com']) wyświetla wartości

for key in contacts: # iterowanie po zmiennej key

    print(key)

Ola

Daniel

Ania

Rafał

for key in contacts.keys(): to samo co wyżej

for key, value in contacts.items():

    print(key, " ", value)

Ola ola@exaple.com

Daniel 30

Ania ania@example.com

Rafał rafał@example.com

# items - użycie wszystkich elementów słownika

contacts.update({"Adam": "brzdąć 1", "Filip": 5, "Karolina": "mama drani"}) # update - doda do słownika contacts nowe pary kluczy i wartości (powiększy słownik o kilka par)

# update może też zmienić wartość , jeśli podany klucz już istnieje w słowniku

print(contacts) # {'Ola': 'ola@exaple.com', 'Daniel': 30, 'Ania': 'ania@example.com', 'Rafał': 'rafał@example.com', 'Adam': 'brzdąć 1', 'Filip': 5, 'Karolina': 'mama drani'}

data = {"name"  : "ola", "city" : "waw"}

dataPostalCode = "postal code"

data[dataPostalCode] = 12345 # dodanie rekordu do słownika 'postal code': 12345

print(data) # {'name': 'ola', 'city': 'waw', 'postal code': 12345}

del data["city"] # {'name': 'ola', 'postal code': 12345}kasowanie elementu słownika

print(data.pop("postal code")) # {'name': 'ola'} kasowanie elementu słownika (2 metoda)

data["city"] = "Rad" # {'name': 'ola', 'city': 'Rad'} dodanie rekordu do słownika 'city': 'Rad'

data.clear() # {} skasowanie wszystkiego w słowniku

data = {"name"  : "kasia", "city" : "Krk"}

dataCopy = data.copy() # utworzenie kopi (płytkiej) słownika data

print(data["name"] is dataCopy["name"]) # True, widzimy, że ten sam element w slowniku data i dataCopy są

                                        # w tym samym miejscu w pamięci (dlatego jest to płytka kopia)

print( data is dataCopy) # False - zmienna data i dataCopy jako słownik są w innym miejscu w pamięci

                         # (w odróżnieniu od elementów zawartych w tych słownikach)

data2 = dict.fromkeys(("name", "city", "code"))# tworzenie nowego słownika po zadeklarowanych kluczach

print(data2) # {'name': None, 'city': None, 'code': None}. Wartości values przyjmują "None"

data3 = dict.fromkeys(("name", "city", "code"), "|MG|" )# tworzenie nowego słownika po zadeklarowanych kluczach

print(data3) # {'name': '|MG|', 'city': '|MG|', 'code': '|MG|'}. Wartości values przyjmują |MG|

print( data2.get("x", "DEFAULT")) # DEFAULT- sprawdzenie, czy slownik data2 ma klucz "x", jeśli nie to DEFAULT

print( data3.get("city", "code doesnt exist")) # |MG|- jeśli tak, to zwróci wartość (value) dla tego klucza

print("name" in data2) # true- sprawdzenie, czy slownik data2 ma klucz "name", TAK- True, NIE- False

Usuwanie zawartości ze slownika(metoda 2). Można użyć też del

print(data3)               # {'name': '|MG|', 'city': '|MG|', 'code': '|MG|'}

print(data3.pop("name"))   # |MG|

print(data3)               # {'city': '|MG|', 'code': '|MG|'}

dict1 = dict(ford = 1950, opel = 1940, mercedes = 1932) #n tworzenie slownika

print(dict1) # {'ford': 1950, 'opel': 1940, 'mercedes': 1932}

dict2 = dict([("ford", 1950), ("opel", 1940), ("mercedes", 1932)]) #n tworzenie slownika [lista(tupli)]

print(dict2) # {'ford': 1950, 'opel': 1940, 'mercedes': 1932}