**String**

# immutable: int, float, bool, star, tuple, frozen set

# Mutable types: list, set, dict

star = "Hello World!"

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

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

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 = star + " 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

print(sorted("AnalityK")) # ['A', 'K', 'a', 'i', 'l', 'n', 't', 'y'] – sortowanie (zwraca listę)

print(sorted("Analityk", reverse=True)) # ['y', 't', 'n', 'l', 'k', 'i', 'a', 'A'] - sortowanie w odwrotnej kolejności

**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}

**Set, frozenset**

setData = {2,3,1,4,5} # set - zbiór. Tworzenie nieuporządkowanego zbióru unikalnych wartości

setData.add(22)       # dodanie wartości do zbioru (add)

setData.discard(1)    # kasowanie elementu ze zbioru (discard), setData.remove(3)     # kasowanie elementu ze zbioru

print(setData)        # {2, 4, 5, 22}

setData.add(5)        # liczba 5 nie została dodana, ponieważ już jest w zbiorze(zbiór zawiera unikalne wartości)

print(1 in setData)   # False - takie zapytanie daje zawsze albo True albo False, print(22 in setData)   # True

setData = {2, 4, 5, 22,}, setData2 = {1,4,5,6,19,20,21,22}

print(setData | setData2) # {1, 2, 4, 5, 6, 19, 20, 21, 22} tworzy z dwóch zbiorów 1 zbiór przy zachowaniu

                          # zasady unikalnych wartości

print(setData & setData2) # {4, 5, 22} tworzy nowy zbiór, w którym użyte będą tylko te wartości, które są zawarte

                          # w obydwu łączonych zbiorach

print(setData2 - setData) # {1, 6, 19, 20, 21} od zbioru 2 odejmujemy zbiór 1

print(setData ^ setData2) # {1, 2, 6, 19, 20, 21} różnica symetryczna - bierze z 1 zbioru wartości nie występujące w 2 zbiorze

                          # oraz z drugiego zbioru wartości nie występujące w pierwszym zbiorze i tworzy z nich nowy zbiór

print(setData.intersection(setData2)) # {4, 5, 22} – tworzy nowy zbiór- tylko wartości zawarte zarówno w setData jak i setData2

print(setData.difference(setData2)) # {2} - wyświetli tylko wartość zbioru setData, ktorej nie ma w zbiorze setData2

print(setData.union(setData2)) # {1, 2, 4, 5, 6, 19, 20, 21, 22} - wyświetli wszystkie wartości występujące w jednym jak i drugim zbiorze(bez powtarzania ich - zunifikuje je)

print(setData.symmetric\_difference(setData2)) # {1, 2, 6, 19, 20, 21} - to samo co print(setData ^ setData2)

frozenData = frozenset(setData) # niemutowalny zbiór wartości

#frozenData.add(22) - polecenie wywoła błąd, ponieważ zbiór jest niemutowalny

**Falsy values**

# falsy values czyli wartości które dają false przy konwersji na boolean

print(bool()) # False

print(bool(False)) # False

print(bool(0)) # False

print(bool(0.0)) # False

print(bool(())) # False- pusta lista

print(bool([])) # False- pusta krotka

print(bool({})) # False- pusty zbiór

print(bool("")) # False- pusty łańcuch znaków

print(bool(None)) # False

print(bool(True)) # True

print(bool(10)) # True- musi być różna od 0 aby było true

print(bool(-10)) # True

print(bool(-12.345)) # True

print(bool((1,2,3))) # True- lista, krotka, zbiór- musi zawierać conajmniej 1 element

print(bool([0])) # True

print(bool({0})) # True

print(bool("z")) # True

**math operators**

a = 12, b = 3

result = a + b; print(result) # 15

result = a - b; print(result) # 9

result = a \* b; print(result) # 36

result = a / b; print(result) # 4.0

result = a % b; print(result) # 0 - modulo, reszta bz dziel

result = a % 9; print(result) # 3 - 12/9=1 dziewięć tylko 1 raz zmieści się w całości w 12, 12-9\*1=3 wynik=3 reszta z d

result = a \*\* b; # 1728 - podnoszenie do potęgi

result = a // b; # 4 tzw. floor division czyli dzielenie dające tylko liczbę całkowitą jako wynik

result = 20 // 3; print(result) # 6 - 20/3=6.666, wynik to tylko liczba calkowita 6

a += 1 # inkrementacja (zwiększanie wartości o 1)

print(pow(2,3)) # 8 – 2 do potęgi 3

# aby policzyć silnie(!) musimy stworzyć funkcję z tzw rekurencją (recursion)

# funkcja rekurencyjna wywołuje samą siebie ze swojego ciała

def silnia(x):

    if x <= 1:

        return 1

    else:

        return x \* silnia(x - 1)

print(silnia(5)) # !5 = 120

# przebieg operacji: 5 \* (5-1=4) \* (4-1=3) \* (3-1=2) \* (2-1=1)

# po podstawieniu za x=5 wykonuje się instrukcja else z zagnieżdżoną kolejną funkcją silnia(x-1).Dopóki x będzie > 1 sytuacja

# będzie się powtarzać . Dopiero w momencie gdy x = 1, zostanie wykonana instrukcja if i zostanie zwrócony wynik końcowy

print( abs(9)) # 9          print( abs(-9.1)) # 9.1

# math.ceil - funkcja zaokrągla do góry (ceil- sufit)

print( math.ceil(11.00000001)) # 12

print( math.ceil(9.999999999)) # 10

print( math.ceil(-1.00000111)) # -1

print( math.ceil(-1.99999999)) # -1

# math.floor - funkcja zaokrągla do dołu (floor - podłoga)

print( math.floor(11.00000001)) # 11

print( math.floor(9.999999999)) # 9

print( math.floor(-1.00000111)) # -2

print( math.floor(-1.99999999)) # -2

# funkcja max/min zwraca największy/najmniejszy z przekazanych argumentów

print(max(-1, 7, 23, 89, 63, -24)) # 89

print(max( [-1, 7, 23, 89, 63, -24] )) # 89

print(max( (-1, 7, 23, 89, 63, -24) )) # 89

print(min(-1, 7, 23, 89, 63, -24)) # -24

print(min( [-1, 7, 23, 89, 63, -24] )) # -24

print(min( (-1, 7, 23, 89, 63, -24) )) # -24

# funkcja pow - podnoszenie do poęgi ( to samo można uzyskać "\*\*") print(pow(4,3)) # 64

# funkcja math.sqrt - wyciąganie pierwiastka kwadratowego print(math.sqrt(1024)) # 32.0

# funkcja round - zaokrąglanie do określonej liczby miejsc po przecinku

print( round(12.7891234, 3)) # 12.789

print( round(12.7891234, 2)) # 12.79

print( round(12.7891234, 1)) # 12.8

# funkcja random - losowy element (z biblioteki random)

print( random.random() ) # losowa liczba zmiennoprzecinkowa od 0 do 0.9999999...

print( random.random() \* 100 ) # losowa liczba zmiennoprzecinkowa od 0 do 9.9999999...

print( int(random.random() \* 100) ) # losowa liczba całkowita od 0 do 99

print( random.randint(1,10)) # losowa liczba całkowita z przedziału od 1 do 10

print( random.choice([0,1,2,3,4,5,6])) # losowy element z listy (krotki, zbioru, łańcucha znaków)

print( random.choice(["ola", "ania", "adam", "filip"])), print( random.choice("abecadło"))

print( random.randrange(-10, 30, 5)) # wartość losowa z zakresu (wartość początkowa -10, wartość końcowa 30, krok 5)

listData = [0,1,2,3,4,5,6,7] random.shuffle(listData) # argumenty listData zostaną zwrócone w losowej kolejności

Wiele funkcji matematycznych dostępnych jest w bibliotece math. Np.:

# Potęgi i logarytmy.

# math.exp(x) - inaczej e \*\* x

# math.log(x) - logarytm naturalny ln(x).

# math.log(x, base) - inaczej math.log(x)/math.log(base)

# math.log1p(x) oznacza math.log(1 + x),lepsze dla małych x

# math.log10(x) - dokładniajsze niż math.log(x, 10).

# math.pow(x, y) - oznacza x \*\* y

# math.sqrt(x) - pierwiastek kwadratowy.

# Funkcje trygonometryczne.

# math.sin(x), math.cos(x), math.tan(x),

# math.asin(x), math.acos(x), math.atan(x),

# math.sinh(x), math.cosh(x), math.tanh(x),

# math.asinh(x), math.acosh(x), math.atanh(x), (Python 2.6)

# math.hypot(x, y)🡪 math.sqrt(x\*x + y\*y)norma euklidesowa.

print ( math.hypot(3, 4) ) # 5.0 float

# math.fabs(x) - wartość bezwzględna float.

# math.radians(x) - zamiana stopni na radiany.

# math.degrees(x) - zamiana radiany na stopnie.

print ( math.pi ) # 3.141592... stała matematyczna

print ( math.e ) # 2.718281... stała matematyczna

# math.factorial(x) - silnia

# math.fsum(iterable)-suma float bez utraty precyzji.

# Nieskończoności: float("inf"), float("-inf"), math.inf

# A floating-point "not a number" (NaN) value:

# float("nan"), math.nan

# math.isinf(x) - float x is positive or negative infinity.

# math.isnan(x) – czy float x is a NaN (not a number).

sum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])

# 0.99999999999999989 gubimy precyzję

math.fsum([.1, .1, .1, .1, .1, .1, .1, .1, .1, .1])

# 1.0 jest dokładniej

# math.erf(x) - the error function.

# math.erfc(x) - the complementary error function.

# math.expm1(x) - dokładniejsze niż math.exp(x)-1.

# math.gamma(x) - the Gamma function.

# math.lgamma(x) - the natural log of the Gamma function.

**assignment operators - operatory przypisania**

a = 12

a += 1 # a = a + 1 12+1=13

a -= 1 # a = a - 1 13-1=12

a \*= 2 # 24

a /= 3 # 8

a %= 3 # 2 - reszta z dzielenia

a \*\*= 4 # 16

a //= 5 # 3 - zwraca dzielenie bez reszty (16/5=3.2)

**comparison operators - operatory porównania**

result = 12 == 8 # czy 12 jest równe 8 - False

print( 12 != 4) # czy 12 nie jest równe 4 - True

print( 8 > 1 ) # True

print( 5 >= 3) # True- wieksze/mniejsze lub równe (<=)

**logical operators (operatory logiczne)- and(i), or(lub) not(nie)**

print( True and True) # True

print( True and False) # False

print( False and True) # False

print( False and False) # False

print ( 10 >= 5 and 3 < 9) # True

print ( 12 < 20 and 5 < 3) # False

print( True or True ) # True

print( True or False ) # True

print( False or True ) # True

print( False or False ) # False

print ( 10 >= 5 or 3 < 9) # True

print ( 12 < 20 or 5 < 3) # True

print( not True) # False

print( not False) # True

print( not( 3 == 3 ) ) # False

print( not( 5 > 10 ) ) # True

print( not( 10 >= 5 and "Ania" != "Ola")) # False

**membership operators - operatory przynależności in, not in**

data = [0,1,2,3,4,5]

print( -1 in data) # Fals - operator in sprawdza,czy wartość jest np. w zmiennej, krotce liście itd.

print( "Ola" in ("Ola", "Ania")) # True

print( 10 not in data) # True - operator not in sprawdza, czy wartość jest np. w zmiennej, krotce liście itd.

print( 2 not in data) # False

**# identity operators - operatory tożsamości is, is not**

# is - operator sprawdza, czy 2 zmienne odnoszą się do tego samego miejsca w pamięci

# is not sprawdza, czy zmienne nie odnoszą sie do tego samego miejsca w pamięci

a = [1,2,3,4,5], b = a 🡪 print( a is b) # True, print( a is not b ) # False

c = [3,4,5]

print( a is c ) # False, print( a is not c) # True

**if, elif, else**

# instrukcja warunkowa if oczekuje wartości true i tylko wtedy uruchomi kod pod nią.  Kod do wykonania

# zwyczajowo wpisuje się po 4 spacjach (nasz edytor VDC sam je robi).

# jeśli wykonujemy po sobie kilka instrukcji if to wykonają się one wszystkie po kolei, a zwrócone zostaną tylko True

# jesli po instrukcjach if występuje elif, to zostanie ona sprawdzona tylko wtedy, gdy, żadna z instrukcji

#   if nie będzie True i nie zostaną one wykonane.

# wykona się tylko pierwsza instrukcja elif = True. Każda kolejna, nawet = True nie zostanie sprawdzona

# po instrukcjach elif możemy sprawdzać instrukcje if. Jeśli = True - zostaną wyświetlone

# instrukcja else zostanie sprawdzona tylko jesli wczesniejsze instrukcje if elif = False. Może ona wystąpić tylko raz

a = 10

if a == 10: # True

    print ("a = 10")    # zostanie wyświetlone

if a > 11: # False

    print("a > 11")     #  nie zostanie wyświetlone

if a != 12: # True

    print("a != 12")    # zostanie wyświetlone

elif a < 14: # True

    print("a < 14")     # nie zostanie wyświetlone

a = 10

if a != 10: # False

    print ("a != 10")    # nie zostanie wyświetlone

if a > 11: # False

    print("a > 11")     #  nie zostanie wyświetlone

if a == 12: # False

    print("a == 12")    # nie zostanie wyświetlone

elif a < 14: # True

    print("a < 14")     # zostanie wyświetlone

elif a != 14: # True

    print("a != 14")     # nie zostanie wyświetlone

if a == 10: # True

    print("a == 10")     # zostanie wyświetlone

elif a > 3: # True

    print("a > 3")      # nie zostanie wyświetlone

if a > 5: # True

    print("a > 5")     # zostanie wyświetlone

else:

    print(a = 10)      # nie zostanie wyświetlone

# uproszczony zapis instrukcji if elif else

print("prawda") if 5 > 2 else print("nieprawda") # prawda

a = "parzysta" if 10 % 2 == 0 else "nieparzysta" # parzysta- przypisanie wyniku instrukcji do zmiennej

**Try, except**

# try,except czyli przechwytywanie wyjątków

x = 12, y = 0

# instrukcje try/except sprawdzają poprawność kodu i w razie znalezienia błędu zwracają określony przez

# programistę komunikat nie przerywając działania programu

try:

    print( x / y )

    print( "działa/nie działa")

except ZeroDivisionError:

    print("niemożliwe jest dzielenie przez 0!!!")

print("dalsze instrukcje\n")

# wynik: niemożliwe jest dzielenie przez 0!!!

dalsze instrukcje

# możliwe jest podanie kilku oczekiwanych błędów do sprawdzenia w kilku instrukcjach except( pierwszy znaleziony błąd przerywa sprawdzanie dalszego kodu)

try:

    print( x + "!") # błąd typów danych

    print( x / y )

    print( "działa/nie działa")

except ZeroDivisionError:

    print("niemożliwe jest dzielenie przez 0!!!")

except TypeEror:

    print("błąd typów danych!!!")

print("dalsze instrukcje\n")

# wynik: błąd typów danych!!!

dalsze instrukcje

# możliwe jest podanie kilku oczekiwanych błędów do sprawdzenia w jednej instrukcji except( pierwszy znaleziony

# błąd przerywa sprawdzanie dalszego kodu)

try:

    print( x + y)

    print( x / y )

    print( x + "!")

    print( "działa/nie działa")

except (ZeroDivisionError, TypeError):

    print("błędny 1 z 3 warunków!!!")

print("dalsze instrukcje\n")

# wynik: 12

błędny 1 z 3 warunków!!!

dalsze instrukcje

# możliwe jest też nie podanie nazwy oczekiwanego błędu. W takim wypadku instrukcja po except zostanie wykonana,

# jeśli pojawi się jakikolwiek error

try:

    lista = []

    print(lista[0])

    print( x + y)

    print( x / y )

    print( x + "!")

    print( "działa/nie działa")

except:

    print("błędny fragment kodu!!!")

finally:

    print("finally wykona się w przypadku wykrycia błędu jak i nie" )

print("dalsze instrukcje\n")

# wynik:

błędny fragment kodu!!!

finally wykona się w przypadku wykrycia błędu jak i nie

dalsze instrukcji

**while loop**

# pętla while- działa dopóki spełniony będzie warunek. Jeśli przestanie być spełniony, skrypt przejdzie do kodu za pętlą,

# 1 # pętla będzie działać aż number osiągnie wartość 0, wtedy się zakończy i skrypt przejdzie do dalszego kodu

number = 5

while number > 0:

    print( number )

    number -= 1

else:               # jednorazowo można użyć instrukcji warunkowej else, która zostanie wykonana po pętli

    print( "number po pętli: " + str(number))

# 2 # tworzenie pętli nieskończonej while

Pętla while True: będzie działać dopóki nie zostanie uruchomiona procedura zakończenia pętli zawarta w jej kodzie.

Jeśli takiej procdury nie będzie, pętla będzie działała w nieskończoność (może to spowodować zawieszenie systemu op.)

num = 0

while True:

    print( "wprowadź liczbę lub exit, aby zakończyć ")

    strData = input() # polecenie input() zażąda wprowadzenia danych z terminala już po uruchomieniu programu

    if strData == "exit" : break # taki zapis umożliwi zaknięcie pętli (break), jeśli wpiszemy w terminalu exit

    num = num + int(strData) # jeśli nie będzie polecenia exit (zakończenie pętli), skrypt w

    print( "Wartość po dodaniu liczby: " + str(num))

**for loop**

# for - pętla,którą możemy wykorzystać do iteracji tylko : listy, krotki, zbioru, słownika, łańcucha znaków

for v in [1,2,3,4]:                 # iteracja listy

    print( v \* 2 )

dictionaryData = { "Ania" : "ania@exaple.com", "Ola" : "ola@example.com", "Jan" : "jan@wp.pl"}

for key in dictionaryData:          # iteracja po kluczu slownika

    print(key)

for key in dictionaryData.keys():   # i to samo inaczej zapisane

    print(key)

for key in dictionaryData.keys():   # przy takim zapisie otrzymamy wartości dla każdego klucza

    print( dictionaryData[key] )

for key, value in dictionaryData.items(): # iteracja po kluczu i wartości slownika jednocześnie

    print( key, " : ", value)

for values in dictionaryData.values(): # iteracja po wartości slownika

for v in "string": # iteracja łąncucha znaków

    print(v)       # otrzymamy wydruk każdej litery naszego string

shopShelf = [1,2,3,4,5,6]

for product in shopShelf: # użycie end = " " powoduje wyświetlenie wyniku tej pętli w jednym wierszu. Poszczególne elementy

print(product, end = " ") # naszej listy są wyświetlane w odstępach spacji (może być to też inny znak) # 1 2 3 4 5 6

data = [0,1,2,3,4,5,6]

for i in data:

    if i == 3:

        break

# wynik działania [0,2,4]- iteracja listy data zostanie przerwana, jeśli i = 3 (zakończy ją funkcja break)

for i in data:

    if i == 3 or i == 5:

        continue

print(i)

[0,1,2,4,6,] # jęśli i=3 albo i=5, pętla for będzie działała, jednak wartość 3 oraz 5 zostaną pominiet i nie zostaną wyświetlone (odpowiada za to funkcja continue)

if 10 > 2:

    pass

else:

pass # funkcja pass nic nie znaczy. Używamy go wtedy, kiedy jakaś instrukcja wymaga dalszego kodu aby nie wygenerować błędu, jednak my nie chcemy go tamwstawiać. W przyszłości

możemy usunąć pass i zastąpić go docelowym kodem f

**funkcje**

def addNumbers(a,b):  # definiujemy funkcę (def), addNumbers - nazwa nowej funkcji, (a,b) - argumenty funcji

    return a + b      # funcja zwróci nam sumę a + b

print(addNumbers(3,4))# 7 - wywołanie funkcji

# domylne wartości funkcji - w funkcji z góry możemy ustalić domyślne argumenty, ktore zostaną wykonane, jeżeli

# odniesienie do naszej funkcji(wykonanie jej) nie będzie ich zawierało

def printCar(brand, name = "concept", year = 1960, color = "black"):

    print(brand, name, year, color)

printCar("Ford") # Ford concept 1960 black - podanie tylko 1 parametru powoduje, że 3 kolejne są dodawane domyślne

printCar("Ford", "Mustang") # wynik: Ford Mustang 1960 black

printCar("Ford", "Mustang", 1970) # wynik: Ford Mustang 1970 black

printCar("Ford", "Mustang", 1970, "red") # wynik: Ford Mustang 1970 red

def fn(a, \*args, \*\*dict\_args): # \*args - możemy wprowadzić dowolną ilość argumentów, zostaną one wyświetlone w postaci krotki

                               # \*\*dict\_args - możemy wprowadzić dowolną ilość kluczy i wartości (dictionary)

    print(a)      # 3

    print(args)   # (6, 8, 9, True, 'cx')

    print(args[0])   # 6

    print(dict\_args)   # {'user': 'admin', 'hasło': 'password'}

fn(3, 6, 8, 9, True, "cx", user = "admin", hasło = "password")

def fn(a, \*args, \*\*dict\_args): # w takiej funkcji możemy argumenty \*args i \*\*dict\_args iterować za pomoća pętli for

  print(a) # 3

for arg in args:

        print(arg)

    for key in dict\_args:

        print(dict\_args[key])

# 6 # cx # admin

# 8 # 9 # password

# True

fn(3, 6, 8, 9, True, "cx", user = "admin", hasło = "password")

def fn(a, b):

    return a + b, a \* b, a - b # instrukcja return zwraca kilka wartości i zostaną one zwrócone w postaci krotki

# pamiętaj, że return konczy działanie funcji i wszystko co będzie poniżej return nie zostanie wykonane

result = fn(3, 4) 🡪 print(result) # (7, 12, -1)

# "/"- wszystkie argumenty przed slashem muszą zostać wprowadzone zgodnie z kolejnością w def funkcji

        # i nie mogą być nazwane. / to parametr pozycyjny

# "\*"- wszystkie argumenty po gwwiazdce muszą zostać wprowadzone jako argumenty nazwane

        # ( i mogą być wprowadzone w różnej kolejności )

def printCar(brand, / ,name = "concept", \* , year = 1960, color = "black"):

    print(brand, name, year, color)

printCar("Ford", "Mustang", year = 1963, color = "blue") # Ford Mustang 1963 blue

printCar("Ford", name = "Mustang", color = "blue", year = 1963) # Ford Mustang 1963 blue

# printCar(brand = "Ford", "Mustang", year = 1963, color = "blue") - błąd (przed "/" parametr nie może być nazwany)

# printCar("Ford", "Mustang", year = 1963, "blue") - błąd (po "\*" parametr musi być nazwany)

def add(x :int = 0, y :int = 0) -> str: # możemy określić podpowiedź jaki typ powinien mieć argument funkcji(otrzymamy podpowiedz w tyrakcie korzystania z funkcji)

    return str(x + y)                   # jeśli wprowadzimy inny typ argumentu, funkcja i tak się wykona

print(add(10, 10)) # 20

print(add.\_\_name\_\_) # add - \_\_name\_\_ - wyświetli nazwę naszej funkcji

print(add) # <function add at 0x00316348> - bezpośrednie odniesienie do naszej funkcji pokaże adres naszej funkcji

def function\_generator(nochange, ifUpper):

    def repeatText(text, how\_many\_repeat = 1):

        return text \* how\_many\_repeat

    def repeatUpper(text :str, how\_many\_repeat = 1):

        return text.upper() \* how\_many\_repeat

    def repeatLower(text :str, how\_many\_repeat = 1):

        return text.lower() \* how\_many\_repeat

    if nochange: # jeżeli nochange = True

        return repeatText # wykona się

    else: # jeżeli nochange = False

        if ifUpper: # jeżeli ifUpper = True

            return repeatUpper # wykona się

        else: # jeżeli ifUpper = False

            return repeatLower # wykona się

functionUpper = function\_generator(False, True)

print(functionUpper("Michał Grabarz ", 2)) # MICHAŁ GRABARZ MICHAŁ GRABARZ

functionLower = function\_generator(False, False)

print(functionLower("Michał Grabarz", 2)) # michał grabarzmichał grabarz

**funkcja lambda (map, filter, reduce)**

# funkcja lambda - jednolinijkowa funkcja bez nazwy

sum = lambda a,b: a + b

print(sum(4,5)) # 9

def generateLambda(num):

    return lambda a: a \* num # funkcja lambda przekazana do naszej definiowanej funkcji generateLambda

doubler = generateLambda(2) # doubler = lambda a: a \* 2

print(doubler(4)) # 8

# kilka sposobów zapisu jednej funkcji liczącej to samo (w tym przypadku kwadrat liczby) z wykorzystaniem lambda

# 1

def funkcja(f,liczba):

    return f(liczba)

print(funkcja(lambda x: x \* x, 3))

# 2

def kwadrat(x):

    return x \* x

print(kwadrat(3))

# 3

wyn = (lambda x: x \* x)(3)

print(wyn)

# 4

wyn = lambda x: x \* x

print(wyn(3))

listData = [0,1,2,3]

result = list( map(lambda a: a \* 3, listData)) # instrukcja map powoduje w tym przypadku przejście po każdym

print(result) # [0,3,6,9]                      # elemencie listData i wykonanie na nim funkcji lambda

def funkcja(a):                                # to samo co wyżej, tylko z użyciem def funkcji

    return a \* 3

result = map(funkcja, listData)

print(list(result)) # [0,3,6,9]

result = list(filter(lambda a: a > 1, listData))# imstrukcja filter powoduje przejście po każdym elemencie listData

print(result) # [2,3]                           # i sprawdzeniu go, czy jest > 1. Jeśli tak, zwraca true, co

                                                # powoduje że przekazywany jest on do naszej nowej listy result

result = reduce( lambda x,y: x + " " + y, ("Ola", "Ania", "baba")) # funkcja reduce redukuje nam w tym momencie listę

print(result) # Ola Ania baba                                      # ("Ola", "Ania", "baba") do łańcucha znaków

# przykłady na filtr imion listy i na sumę elementów listy

print("\n inne przykłady \n ")

listData = ["ola", "włodzimierz", "benedykt", "kasia", "ania", "jarosław"]

result = filter( lambda x: len(x) <= 5, listData)# filtruje imiona i zwraca te, kóre mają 5 lub mniej liter

print(list(result)) # ['ola', 'kasia', 'ania']

from functools import reduce

numSum = reduce( lambda x,y: x + y, [0,1,2,3,4,5,6,]) # wynik będzie sumą elementów listy [0,1,2,3,4,5,6,]

print(numSum) # 21

print(type(numSum))

**date time function**

import time, import datetime #import bibliotek/modułów dotyczących czasu i daty

ticks = time.time() # czas, jaki upłynął od 1 stycznia 1970 roku podawany w sekundach (stempel czasu)

print(ticks) # 1666770302.785295

timeData = time.localtime() # wynikiem będzie krotka(tzw nazwana krotka) z aktualną datą i czasem

print(timeData) # time.struct\_time(tm\_year=2022, tm\_mon=8, tm\_mday=16, tm\_hour=22, tm\_min=56, tm\_sec=59,

# tm\_wday=1, tm\_yday=228, tm\_isdst=1)

# tm\_wday=1 - dzień tygodnia(0 - poniedziałek, 1 - wtorek itd.)

# tm\_yday=228 - 228 dzień roku

# tm\_isdst=1 - czy python zarządza czasem letnim(1), czy zimowym(0)

print(timeData.tm\_year) # 2022 - pobieramy z krotki np. rok

timeData = time.localtime(10) # przekazanie stempla czas( krotka będzie zawierać dane z 10 sekund

# od rozpoczęcia 1 stycznia 1970 roku)

# time.struct\_time(tm\_year=1970, tm\_mon=1, tm\_mday=1, tm\_hour=1, tm\_min=0, tm\_sec=10, tm\_wday=3,

# tm\_yday=1, tm\_isdst=0)

result = time.asctime( time.localtime()) # wyświetlenie daty i godziny w bardziej przystępny sposób

print(result) # Wed Oct 26 09:45:02 2022

timeData = time.localtime()

print(time.strftime("%d/%m/%Y %H:%M:%S",timeData)) # 26/10/2022 09:45:02 - formatowanie daty/godziny według upodobań

# %d/%m/%Y %H:%M:%S - day, month, year, hours, minutes, seconds)(znaki / : wstawione między %d zostaną pokazane)

timeStr = "12:10:45 12.10.1984" # wprowadzamy sami datę i godzinę

timeData = time.strptime(timeStr, "%H:%M:%S %d.%m.%Y") # przeformatowanie daty  do krotki (taką jak dla time.localtime())

i = 0

while i < 12:

    time.sleep(0.1) # funkcja sleep usypia działanie pętli o podaną wartość (w tym przypadku o 0,1 s)

    print(time.asctime(time.localtime())) # będzie nam wyświetlało co 0.1 s datę i godzinę

    i += 1

tStart = time.perf\_counter() # czas początku wykonywania np.kodu

time.sleep(0.2)

tEnd = time.perf\_counter() # czas końca wykonywania np.kodu

print("Code took: ", (tEnd - tStart), "seconds") # wynikiem będzie czas wykonania kodu time.sleep(1.2)

dateTimeObj = datetime.datetime.now() # zwraca aktulną datę i czas

print("format: ", dateTimeObj.strftime("%H:%M:%S %d.%m.%Y")) # 22:45:45 10.03.2025-formatowanie zmiennej na bardziej przystępną

dateTimeObj = datetime.datetime(2025, 3, 10)

dateTimeObj = datetime.datetime(2025, 3, 10, 22, 45, 45)

print("date(): ", dateTimeObj.date()) # date():  2025-03-10

print("time(): ", dateTimeObj.time()) # time():  22:45:45

print("timestamp(): ", dateTimeObj.timestamp()) #znak czasu

print("today(): ", dateTimeObj.today())#akt. data i godzina

print("year: ", dateTimeObj.year) # aktualny rok

print("month: ", dateTimeObj.month) # aktualny miesiąc

print("day: ", dateTimeObj.day) # aktualny dzień

print("hour: ", dateTimeObj.hour) # aktualny godzina

print("minute: ", dateTimeObj.minute) # aktualna minuta

print("second: ", dateTimeObj.second) # aktualna sekunda

print("format: ", dateTimeObj.strftime("%H:%M:%S %d.%m.%Y")) # 22:45:45 10.03.2025-formatowanie zmiennej na bardziej przystępną

dateTimeObj = dateTimeObj.now()

print("format: ", dateTimeObj.strftime("%H:%M:%S %d.%m.%Y")) # 00:20:50 18.08.2022-formatowanie zmiennej na bardziej przystępną

# porównywanie dat

dateTime1 = datetime.datetime( 2025, 1, 1, 23,59,59)

dateTime2 = datetime.datetime( 2030, 1, 1, 23,59,59)

print(dateTime1 < dateTime2) # True

print(dateTime1 > dateTime2) # False

date1 = datetime.date(2025,1,1)

date2 = datetime.date(2027,1,1)

print(date1 < date2) # True

print(date1 > date2) # False

print(date2 - date1) # 730 days, 0:00:0

**Bubble sort(sortowanie bąbelkowe**)

# poniższa lista sortuje argumenty listy od najmniejszej do największej (liczby, litery, słowa)

def sortowanie\_babelkowe(lista):

    n = len(lista)

    while n > 1:

        zamien = False

        for v in range(0, n-1):

            #print(lista) #- zobaczysz jak to działa

            if lista[v] > lista[v+1]:

                lista[v], lista[v+1] = lista[v+1], lista[v]

                zamien = True

        n -= 1

        if zamien == False: break  --> return print(lista)

sortowanie\_babelkowe(["ac", "aa", "az", "azz", "azb", "abc", "aba"]) # wynik ['aa', 'aba', 'abc', 'ac', 'az', 'azb', 'azz']

print(sorted(["ac", "aa", "az", "azz", "azb", "abc", "aba"])) # ['aa', 'aba', 'abc', 'ac', 'az', 'azb', 'azz'] to samo z sorted

**generator yield**

from tkinter import N

def gen():

    i = 0

    while i < 5:

        yield i # generator yield - zwraca wynik funkcji while, ale w odróżnieniu od return nie przerywa działania funkcji

        i += 1

for i in gen(): # taka iteracja po zdefiniowanej przez nas funkcji jest możliwa tylko, gdy w definiowanej

    print(i)    # funcji użyjemy pętli z generatorem yield

print(list(gen())) # [0, 1, 2, 3, 4] - chcąc wyświetlić wynik funkcji gen musimy dodać ją do jakiejś kolekcji (np listy). W innym razie print wyświetli informację o przechowaniu w pamięci wyniku tej funkcji <generator object gen at 0x000001C660301EE0>

def parzyste(x):

    i = 0

    while i <= x:

        if i % 2 == 0: #sprawdzi czy i parzyste

            yield i

        i += 1

for i in parzyste(16):

    print(i)

print(list(parzyste(16)))

# [0, 2, 4, 6, 8, 10, 12, 14, 16]

def number\_generator(end):

    n = 1

    while n < end:

        yield n

        n += 1

values = number\_generator(25)

print(next(values)) # 1 - aby wywołać generator stosujemy funkcję next. Każde jej użycie generuje wykonanie jednego obejścia i generuje 1 liczbę z naszego zakresu funkcji number\_generator

print(next(values)) # 2 it

**dekoratory**

def decorator(func): # tworzymy naszą funkcję z dekoratorem

    def wrapper():

        print("------------")

        func()

        print("------------")

    return wrapper

def hello(): # tworzymy funkcję

    print("hello world")

hello = decorator(hello) # tworzymy obiekt na naszej def decorator

hello() # ------------

        # hello world

        # ------------

@decorator  # można też tak odnieść się do naszego dekoratora

def witaj():

    print("witaj świecie")

witaj() # ------------

        # witaj świecie

        # ------------

**Dekoratory (statyczne, klasowe)**

from klasa10dekoratory\_statyczneklasowe import Numbers

from klasa101dekoratory\_statyczneklasowe import Animal

przykład = Numbers() # tworzymy obiekt (klasa Numbers)

przykład.addNumber(3)  # korzystając z def add.number() dodajemy elementy do naszej listy self.numbers = numbers

przykład.addNumber(5) przykład.addNumber(8)

przykład.addNumber(6) przykład.addNumber(13)

przykład.addNumber(7) przykład.addNumber(2)

print(przykład.sumNumbers()) # 44 - z def z klasy Numbers

print(przykład.multiNumbers()) # 131040

print(Numbers.subtractNumbers(15, 5)) # 10 metoda statyczna korzystamy z klasy pomijając tworzenie instancji(obiektu)

Numbers.printInformation() # metoda klasyczna możę pobierać dane z naszej klasy nasz printInformation() ma cls.my\_numbers +=1 więc dodaje 1 do pola my\_numbers po wywoł. Metody klasycznj printInformation() każde odniesienie do my\_numbers będzie +1

print("Pole w klasie:") print("po @classmethod:")

print(przykład.my\_numbers) # 10 print(przykład.my\_numbers) # 11

print(Numbers.my\_numbers) # 10 print(Numbers.my\_numbers) # 11

kot1 = Animal(Animal,"Łapek") # Tworzymy obiekt. Ponieważ metoda klasowa ma wbudowany inicjalizator( a nie klasa głowna)

kot2 = Animal(Animal,"Łapek") # tworząc obiekt na klasie Animal, musimy dla naszej  @classmethod podać rownież, że korzysta ona z klasy Animal

kot3 = Animal(Animal,"Łapek")

kot4 = Animal(Animal,"Łapek")

kot5 = Animal(Animal,"Łapek")

print(kot5.howManyAnimal) # metoda cls.howManyAnimal po każdym utworzonym obiekcie będzie zwiększała o 1 nasze pole(zmienną) howManyAnimal znajdującą się bezpośrednio w naszej klasie

class Numbers:

    my\_numbers = 10

    def \_\_init\_\_(self, numbers = []):

        self.numbers = numbers

    def sumNumbers(self): # funkcja ma zsumować wszystkie

        elementy listy self.numbers

summary = 0

        for x in self.numbers:

            summary += x

        return summary

    def multiNumbers(self): # przemnoży przez siebie wszystkie

        elementy listy self.numbers

product = 1

        for x in self.numbers:

            product \*= x

        return product

    def addNumber(self, number): # funkcja pozwoli dodać

        argumenty do listy self.numbers

self.numbers.append(number)

    @staticmethod # tworzymy funkcję(metodę) statycznie.

Możemy z niej korzystać bez tworzenia instancji(obiektu)

    def subtractNumbers(a,b):

        return a - b

    @classmethod # metoda klasowa może uzyskać dane z naszej

klasy (w odróżnieniu od @staticmethod )

    def printInformation(cls): # (cls) - musi być w metodzie,

żeby mogła uzyskać dane z naszej klasy

        print("Jestem class method")

        print ("na rzecz klasy", cls.\_\_name\_\_) # cls.\_\_name\_\_ -

wyświetli nazwę naszej klasy

        cls.my\_numbers +=1 # w metodzie klasycznej możemy odnosić się do pól z naszej klasy np. my\_numbers i je zmieniać

class Animal:

    howManyAnimal = 0

    @classmethod

    def \_\_init\_\_(cls, self, name\_animal): # metoda klasowa z

wbudowanym inicjalizatorem

        self.name\_animal = name\_animal

        cls.howManyAnimal += 1

**wyrażenia regularne)(match,search,findall,sub)**

import re

wzor = r"banan\nbanan\tbanan" # banan\nbanan\tbanan # r - row czyli surowy(nie będzie czytało znaków specjalnych \n \t)

wzor = r"banan" tekst = r"gruszkabananjabłko" tekst1 = r"banangruszkabananjabłko"

print(re.match(wzor, tekst)) # None # match(dopasuj) - domślnie szuka 1 argument w 2 argumencie na jego początku.

print(re.match(wzor, tekst1)) # <re.Match object; span=(0, 5), match='banan'>

print(re.match(r".\*" + wzor + r".\*", tekst)) # zastosowanie r".\*" powoduje, że match szuka w całym tekście

                                             # <re.Match object; span=(0, 18), match='gruszkabananjabłko'>

print(re.search(wzor, tekst)) # search(szukaj) -przeszukuje cały tekst i znajduje szukany wyraz nawet w środku

                              # <re.Match object; span=(7, 12), match='banan'>

wzor = r"banan"

tekst1 = r"gruszkabananjabłkogruszkabananjabłkogruszkabananjabłko"

print(re.findall(wzor, tekst1)) # ['banan', 'banan', 'banan']findall- znajduje wszystkie dopasowania i zwraca w postaci listy

wzor = r"banan" tekst = r"gruszkabananjabłko"

dopasowanie = re.search(wzor, tekst)# jeśli nasze wyrażenie przypiszemy do zmiennej, zachowuje się ono jak obiekt

if dopasowanie:

print(dopasowanie.group()) # banan - group wskazuje wszystkie grupy, które udało się dopasować

print(dopasowanie.start()) # 7 –[id] początek szukanego słowa

print(dopasowanie.end()) # 12 – [id] koniec szukanego słowa

print(dopasowanie.span()) # (7, 12) -krotka z początkiem i końcem szukanego słowa w stringu (span -łącznik)

wzor = r"banan" tekst = r"gruszkabananjabłko"

tekst2 = re.sub(wzor, "JAGODA", tekst) # sub - wyszukuje nasz wyraz(argument 1 - wzor) w naszym tekście

                                       #(arg 3 - tekst) i zastępuje go nowym wyrazem (arg 2 - "JAGODA")

if re.match("ko.", "kot"): # kropka zastępuje każdy znak, więc konsola wyświetli, że arg1 dopasowano do arg2

    print("Dopasowano!")

else: print("Nie dopasowano!")

if re.match("^ko.$", "kotttt"): # Nie dopasowano ^ -pokazuje, gdzie ma się zaczynać dany wzór, $ - gdzie ma się kończyć

    # Nie dopasowano - przeszukiwany tekst nie kończy się jak we wzorze (np. kot, kos, kop)

if re.match("^[Kk]o.$", "lot"): # Nie dopasowano [Kk] lub [pk] - tzw. klasa znaku - będzie szukało w tekśccie liter

    # z nawiasu[]. Jeśli jakaś z nich będzie w naszym tekście, to: Dopasowano!

if re.match("^[Rr]ok[-\_=][0-9][0-9][0-9][0-9]$", "Rok-1984"): # Dopasowano

if re.match("^[A-Z][a-z]a\*$", "Alaaa"): # Dopasowano # \* - gwiazdka stojąca za znakiem/klasą znaków/literą dopuszcza wystąpienie

                                      # jej nieskończoną ilość razy lub niewystąpienie jej wcale

if re.match("^[A-Z][a-z]a+$", "Al"): # Nie dopasowano! # plus działa podobnie jak gwiazdka z tą różnicą, że znkak za którym

# stoi musi wystąpić conajmniej 1 raz(może nieskończoną ilość razy)

if re.match("^[A-Z][a-z]?[A-Z]$", "AA"): # Dopasowano! # ? - znak, za którym stoi może nie wystąpić wcale lub wystąpić 1 raz

if re.match("^[A-Z][a-z]{2,5}$", "Ala"): # Dopasowano! # {2,5} - znak, za którym stoi może wystąpić od 2 do 5 razy

# re.match(wzór do szukania, tekst do przeszukania)

# ^ -pokazuje, gdzie ma się zaczynać dany wzór

# $ - gdzie ma się kończyć dany wzór

# ko. - kropka zastępuje każdą literę, wyrazy np. kot, koc, koń będą dopasowane poprawnie

# [Kk] - szuka w tekście liter Wielkie i małe K

# [A-Z] - szuka w tekście liter z przedziału A-Z (wielkie litery)

# [A-Za-z] - szuka w tekście liter z przedziału A-Z (wielkie i małe litery)

# [^A-Ca-c] - ^ - zastosowanie w klasie znaku ^ powoduję negację danego zakresu liter. W tym

#           przypadku będzie szukało w tekście wszystkich liter z poza zakresu małych i wielkich A-C

# [A-Z][a-z]a\* - \* - gwiazdka stojąca za znakiem/klasą znaków/literą dopuszcza wystąpienie

#                    jej nieskończoną ilość razy lub niewystąpienie jej wcale

# [A-Z][a-z]a+ - + - plus działa podobnie jak gwiazdka z tą różnicą, że znkak za którym stoi musi wystąpić

#                    conajmniej 1 raz(może nieskończoną ilość razy)

# ^[A-Z][a-z]?[A-Z]$ - ? - znak, za którym stoi może nie wystąpić wcale lub wystąpić 1 raz

# ^[A-Z][a-z]{2,5}$ - znak, za którym stoi może wystąpić od 2 do 5 razy

# ^[A-Z][a-z]{2,}$ - znak, za którym stoi może wystąpić od 2 nieskończonej ilości razy

# ^[A-Z][a-z]{,5}$ - znak, za którym stoi może wystąpić od 0 do 5 razy

Przykłady:

re.match("ko.", "kot")  # D

re.match("ko.", "koc")  # D

re.match("ko.", "ko$")  # D

re.match("ko.", "ko9")  # D

re.match("ko.", "ko ")  # D

re.match("^ko.$", "kotttt")  # Nie D

re.match("^ko.$", "kot")  # D

re.match("^[Kk]o.$", "lot")  # Nie D

re.match("^[LlKk]o.$", "lot")  # D

re.match("^[LlKk]o.$", "Lot")  # D

re.match("^[LlKkp]o.$", "pot")  # D

re.match("^[a-z]o.$", "pot")  # D

re.match("^[A-Z]o.$", "pot")  # Nie D

re.match("^[A-Za-z]o.$", "pot")  # D

re.match("^[A-Za-z]o.$", "Pot")  # D

re.match("^[^A-Za-z]o.$", "Pot")  # Nie D

re.match("^[^A-Ka-k]o.$", "Pot")  # D

re.match("^[^A-Za-z]o.$", "$ot")  # D

re.match("^[Rr]ok[-\_=][0-9][0-9][0-9][0-9]$", "Rok-1984")  # D

re.match("^[Rr]ok[-\_=][0-9][0-9][0-9][0-9]$", "Rok-984")  # Nie D

re.match("^[Rr]ok[-\_=][0-9]\*$", "Rok\_984")  # D

re.match("^[Rr]ok[-\_=][0-9]\*$", "rok\_145634984") # D

re.match("^[Rr]ok[-\_=][0-9]\*$", "rok\_")# D

re.match("^[A-Z][a-z]a\*$", "Al")  # D

re.match("^[A-Z][a-z]a\*$", "Ala") # D

re.match("^[A-Z][a-z]a\*$", "Alaaa") # D

re.match("^[A-Z][a-z]a\*$", "Alc")# Nie D

re.match("^[A-Z][a-z]a\*$", "Alan")# NieD

re.match("^[A-Z][a-z]a\*$", "Al") # D

re..match("^[A-Z][a-z]a\*$", "al") # Nie D

rematch("^[A-Z][a-z]a+$", "Ala")  # D

re.match("^[A-Z][a-z]a+$", "Alaaaa") # D

re.match("^[A-Z][a-z]a+$", "Al")  # Nie D

re.match("^[A-Z][a-z]?[A-Z]$", "AA") # D

re.match("^[A-Z][a-z]?[A-Z]$", "AdA")# D

re.match("^[A-Z][a-z]?[A-Z]$", "AddA")# Nie D

re.match("^[A-Z][a-z]{2,5}$", "Ala") # D

re.match("^[A-Z][a-z]{2,5}$", "Alabcd")# D

re.match("^[A-Z][a-z]{2,5}$", "Al")# NieD

re.match("^[A-Z][a-z]{2,}$", "Sebastian")  # D

re.match("^[A-Z][az]{,5}$","Sebastian")# Nie D

re.match("^[A-Z][a-z]{,5}$", "A")  # D

wynik = re.match(r"^(Hello) (Wor(ld))$", "Hello World") # zastosowanie () we wzorze tworzy tzw. grupę (nienazwaną)

                                                        # tworząc grupy możemy je w sobie zagnieżdżać

if wynik:

    print("Dopasowano!")

    print(wynik.group()) # Hello World

    print(wynik.group(0)) # Hello World

    print(wynik.group(1)) # Hello

    print(wynik.group(2)) # World

    print(wynik.group(3)) # ld

    print(wynik.groups()) # ('Hello', 'World', 'ld') # wyświetla wszystkie grupy

wynik = re.match(r"^(Hello) (World)+$", "Hello WorldWorld") # plus stojący przed grupą powoduje, że musi ona

# wystąpić w przeszukiwanym tekście conajmniej 1 raz, ale może nieskończenie wiele razy

    print(wynik.group()) # Hello WorldWorld

wynik = re.match(r"^((?:He)(?P<first>ll)o) (World)$", "Hello World") # (?P<first>ll) - grupa nazwana (nazwa : first)

# jakbyśmy stworzyli we wzorze taką grupę (?:He) - nię będzie ona indeksowana (nie zostanie też wyświetlona)

if wynik:

    print("Dopasowano!")

    print(wynik.group()) # Hello World

    print(wynik.group(0)) # Hello World

    print(wynik.group(1)) # Hello

    print(wynik.group(2)) # ll

    print(wynik.group(3)) # World

    print(wynik.groups()) # ('Hello', 'll', 'World')

    print(wynik.group("first")) # ll - do grupy nazwanej możemy się odnieść za pomocą jego nazwy (first)

wynik = re.match(r"^((?:He)(?P<first>ll)o)( World)+(!|\.)$", "Hello World World World.") # (!|\.) na końcy tekstu

# ma wystąpić albo ! albo "." Przed kropką musi być \ (inaczej kropka by nam wskazywała na każdy znak (!|.))

# | - oznacza lub

if wynik:

    print("Dopasowano!")

    print(wynik.group()) # Hello World World World.

    print(wynik.group(0)) # Hello World World World.

    print(wynik.group(1)) # Hello

    print(wynik.group(2)) # ll

    print(wynik.group(3)) # World

    print(wynik.groups()) # ('Hello', 'll', ' World', '.')

    print(wynik.group("first")) # ll

# walidacja adresu email, poniższe wyrażenie sprawdza, czy adres email jest prawidłowy

if re.match(r"^([A-Za-z0-9]+|[A-Za-z0-9][A-Za-z0-9\.-]+[A-Za-z0-9])@([A-Za-z0-9]+|[A-Za-z0-9][A-Za-z0-9-\.]+[A-Za-z0-9])\.[A-Za-z0-9]+()$", "grabarz.michal@gmail.com"):

**Wyrażenia listowe**

lista = [10] \* 12 # [10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10]

# kolekcje

napis = "abracadabra" \* 3

print(napis) # abracadabraabracadabraabracadabra

lista2 = [10 for i in range(10)]

print(lista2) # [10, 10, 10, 10, 10, 10, 10, 10, 10, 10]

# lista [wyrażenie for element in kolekcja]

lista3 = [litera for litera in "Michał Grabarz"]

print(lista3) # ['M', 'i', 'c', 'h', 'a', 'ł', ' ', 'G', 'r', 'a', 'b', 'a', 'r', 'z']

lista4 = [int(liczba) for liczba in "0123456789"]

print(lista4) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

lista5 = [int(liczba) for liczba in input("Podaj liczby oddzielone spacją: ").split()] #Podaj liczby oddzielone spacją: 3 5 7 9

print(lista5) # [3, 5, 7, 9]

x, y = [int(liczba) for liczba in input("Podaj liczby oddzielone spacją: ").split()] # Podaj liczby oddzielone spacją: 3 94

print(f"x: {x}, y: {y}") # x: 3, y: 94

slownik = {x: x\*x for x in range(11)}

print(slownik) # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}

lista = [[x for x in range(3)] for y in range(5)]

print(lista) #  [[0, 1, 2], [0, 1, 2], [0, 1, 2], [0, 1, 2], [0, 1, 2]]

lista = {y: [x for x in range(3)] for y in range(5)} # po modyfikacji stworzy słownik

print(lista) #  {0: [0, 1, 2], 1: [0, 1, 2], 2: [0, 1, 2], 3: [0, 1, 2], 4: [0, 1, 2]}

lista3 = [litera.lower() for litera in "Michał Grabarz"] # tylko małe litery

print(lista3) # ['m', 'i', 'c', 'h', 'a', 'ł', ' ', 'g', 'r', 'a', 'b', 'a', 'r', 'z']

lista4 = [x+y for x,y in ([2,2], [3,3])]

print(lista4) # [4, 6]

lista5 = [x+y for x,y in ([a,a+1] for a in range(10))]

print(lista5) # [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

parzyste = [x\*x for x in range(20) if x % 2 == 0]

print(parzyste) # [0, 4, 16, 36, 64, 100, 144, 196, 256, 324]

napis = "Michał Grabarz telefon +48739093883"

lista6 = [int(znak) for znak in napis if znak.isdigit()] # isdigit()- wyłapuję tylko ciągi liczb i je zwraca

print(lista6) # [4, 8, 7, 3, 9, 0, 9, 3, 8, 8, 3]

**Operacje na plikach**

plik = open("d:\\python\\basics\\inne\\test.txt", "a") # utworzy plik na podanej przez nas ścieżce pliku.

# podanie tylko samej nazwy pliku(plik = open("test.txt", "w")) utworzy nam plik w katalogu głównym (basics)

# drugi argument oznacza:

# w (write) - plik do zapisu, (za każdym razem otwarcie pliku w formie do zapisu powoduje wyzerowanie

# poprzedniej zawartości tego pliku)

# r (read) - plik do odczytu,

# a (append) dołączać - pozwoli na dołączenie do pliku dalszego tekstu bez usuwania starego

if plik.writable():

    ile = plik.write(input("wprowadź tekst: ") + "\n")

    print("Zawartość: ", ile) # zmienna utworzona na funkcji

write zwraca nam ilość znaków wprowadzonego tekstu

# np. michal grabarz- Zawartość: 15 (pamietaj o znaku zmiany linii \n)

plik.close()

plik = open("d:\\python\\basics\\inne\\test.txt", "r")

if plik.readable():

    tekst = plik.read() # odczytanie pliku do zmiennej tekst

    print("\nZawartość pliku:\n" + tekst)

plik.close()

plik = open("d:\\python\\basics\\inne\\test.txt", "r")

if plik.readable():

    tekst = plik.readlines() # tworzy listę z poszczeglnych

wierszy w pliku txt

    print(tekst)

    for v in tekst:

        print(v)

plik.close()

wprowadź tekst: michal grabarz

Zawartość: 15

Zawartość pliku:

michal

karolina

filip

adam

michal

mich

gregr

michal grabarz

['michal\n', 'karolina\n', 'filip\n', 'adam\n', 'michal\n', 'mich\n', 'gregr\n', '\n', 'michal grabarz\n']

michal

karolina

filip

adam

michal

mich

gregr

michal grabarz

**Analiza tekstu**

plik = open("d:\\python\\basics\\inne\\analizatekstu.txt", "r")

tekst = plik.read() # odczytujemy zmienną z tekstem do analizy (wykorzystano przykładowy tekst)

plik.close()

def policz(txt, znak): # definicja pozwala nam policzyć ilość wystąpień danego znaku w tekscie

    licznik = 0

    for z in txt:

        if z == znak:

            licznik += 1

    return licznik

print(policz(tekst, "a")) # 232 - wyświetla ilość danej liter, w tym przypadku a

print(policz(tekst, "A")) # 86 - wyświetla ilość danej liter, w tym przypadku A

print(policz(tekst, "a") + policz(tekst, "A")) # 318 - wyświetla ilość danej litery, a małe + A duże

print(policz(tekst.lower(), "a")) # 318 - suma a + A (to samo co wyżej, tylko uproszczony zapis - lower)

for z in "abcdefghijklmnoprstuwxyz,.-+=:;@!#$%":

    ile = policz(tekst.lower(), z)  # lower - powoduje zamianę wszystkich liter w tekście na małe, dzięki

# czemu otrzymujemy ilość wystąpień każdej z liter bez względu na to # # czy jest duża czy mała

    procent = 100 \* ile / len(tekst)

    print("{0} - {1} - {2}%".format(z.upper(), ile, round(procent, 2)))

# powyżej sprawdzamy cały tekst i otrzymujemy wynik dla każdej litery z ciągu znaków

# ( który jest tak naprawdę listą) w postaci ilości wystąpień i procentowego udzialu w całym tekscie

lista = ("the", "always", "install", "you", "has", "is", "michal")

for z in lista:

    tekst = tekst.lower()

    ile = tekst.count(z)

    print("{0} - {1}".format(z, ile))

# sprawdza ilość wystąpień poszczególnych wyrazów w tekście (bez względu na wielkość liter(lower))

A - 318 - 5.88%

B - 47 - 0.87%

C - 132 - 2.44%

D - 147 - 2.72%

E - 491 - 9.08%

F - 149 - 2.76%

G - 68 - 1.26%

H - 122 - 2.26%

I - 374 - 6.92%

J - 5 - 0.09%

K - 28 - 0.52%

L - 138 - 2.55%

M - 101 - 1.87%

N - 283 - 5.23%

O - 412 - 7.62%

P - 91 - 1.68%

R - 339 - 6.27%

S - 325 - 6.01%

T - 425 - 7.86%

U - 109 - 2.02%

W - 83 - 1.53%

X - 13 - 0.24%

Y - 103 - 1.9%

Z - 3 - 0.06%

, - 45 - 0.83%

. - 44 - 0.81%

- - 4 - 0.07%

+ - 0 - 0.0%

= - 0 - 0.0%

: - 11 - 0.2%

; - 0 - 0.0%

@ - 3 - 0.06%

! - 13 - 0.24%

# - 0 - 0.0%

$ - 0 - 0.0%

% - 1 - 0.02%

the - 65

always - 1

install - 2

you - 19

has - 3

is - 56

michal - 2

**Klasy**

# klasa jest to zbiór zmiennych i funcji (w klasach nazywanych metodami), które tworzą gotowy szablon rozwiązań, które możemy wykorzystać do stworzenia np. obiektu . OPP - object oriented programming

obiekt = czlowiek() # tworzymy odwołanie do naszej klasy czlowiek

print(obiekt.imie) # sebastian ( korzystamy ze zdefiniowanej przez nas metody

# imie)

print(obiekt.przedstawSie()) # Cześć, mam na imie sebastian ( korzystamy ze

# zdefiniowanej przez nas metody przedstawSie)

print(type(obiekt)) # <class '\_\_main\_\_.czlowiek'>

obiekt2 = czlowiek()

obiekt2.imie = "Adrian" # zmieniamy imie dla nowego obiektu obiekt2

print(obiekt2.przedstawSie()) # Cześć, mam na imie Adrian

print(obiekt.przedstawSie()) # Cześć, mam na imie sebastian

class czlowiek:

    imie = "sebastian"

    def przedstawSie(self):

        return "Cześć, mam na imie " + self.imie

# zmienne i definicje funkcji zawarte w klasie stają się metodami dla obiektów, które utworzymy na ich bazi

obiekt = czlowiek1()

print(obiekt.przedstawSie()) # Cześć , mam na imie sebastian

obiekt2 = czlowiek1()

obiekt2.imie = "Adrian" # nadpisujemy imie dla obiekt2

print(obiekt2.przedstawSie()) # Cześć , mam na imie Adrian

print(obiekt.przedstawSie("Witaj")) # Witaj , mam na imie

# sebastian

print(obiekt2.przedstawSie("Dzień dobry")) # Dzień dobry , mam

# na imie Adrian

class czlowiek1:

    imie = "sebastian"

    def przedstawSie(self, powitanie = "Cześć"):

        return powitanie + " , mam na imie " + self.imie

# w tym przypadku tworzymy zmienną powitanie, do ktorej będzie się odnosiła zdefiniowana przez nas funkcja przedstawSie

# NALEŻY PAMIETAĆ, ŻE DO PIERWSZEGO ARGUMENTU W NASZEJ ZDEFINIOWANEJ FUNKCJI(self) NIE MOŻEMY SIĘ ODNIEŚĆ. JEST ON TAK

# JAKBY ZABLOKOWANY. DO KAŻDEGO KOLEJNEGO JUŻ MOŻEMY TO ZROBIĆ PAMIĘTAJĄC, ŻE ODNOSIMY SIĘ DO NIEGO JAKBY BYŁ PIERWSZY

# jeżeli korzystamy  w klasie z konstruktora, to tworzony w oparciu o klasę obiekt musi wykorzystać wszystkie zmienne (bez self) zawarte w tym konstruktorze (innaczej wystąpi błąd)

obiekt = czlowiek2("Adam", 2)

print(obiekt.przedstawSie())

obiekt2 = czlowiek2("Filip", 5)

print(obiekt2.imie) # Filip

print(obiekt2.wiek) # 5

print(obiekt2.przedstawSie())

print(obiekt.przedstawSie("Witaj"))

print(obiekt2.przedstawSie("Dzień dobry"))

# specjalna metoda inicjalizacji danych (konstruktor). Obiekt powstały w oparciu klasę z konstruktorem \_\_init\_\_ przejmuje jego parametry (bez self)

class czlowiek2:

    def \_\_init\_\_(self, imie, wiek):

        self.imie = imie # zmienna self.imie (self odnosi się

#do pierwszego argumentu def \_\_init\_\_) # to zmienna globalna dla całej klasy, natomiast imie do zmienna lokalna dla def \_\_init\_\_

        self.wiek = wiek

    def przedstawSie(self, powitanie = "Cześć"):

        return powitanie + " , mam na imie " + self.imie + " lat " + str(self.wiek) + "

**Klasy dziedziczenie**

dog = Dog("Reksio", 10) # tworzymy obiekt oparty na klasie Dog, jednak ma on też dostępne metody

                        # (name, age) z klasy bazowej Animal

print(dog.name) # Reksio

print(dog.age) # 10

dog.voice() # How How

cat = Cat("Klakier", 7)

print(cat.name) # Klakier

print(cat.age) # 7

cat.getVoice() # Meow Meow

wolf = Wolf("Geralt", 25)

print(wolf.name)# Geralt

print(wolf.age) # 25

wolf.getVoice() # Jestem wilkiem, How How

class Animal:  # klasa bazowa (rodzic)

    def \_\_init\_\_(self, name, age):

        self.name = name

        self.age = age

class Dog(Animal):# klasa dziedzicząca dziecko od klasy animal

    def voice(self):

        print("How How")

class Wolf(Dog):# klasa wolf dziedziczy od

# klasy Dog (i przez klasę Dog od klasy Animal)

    def getVoice(self):

        print("Jestem wilkiem, ")

        super().voice()             # super - funkcja przeszukuje klasę bazową w poszukiwaniu metody (voice)

class Cat(Animal):

    def getVoice(self):

        print("Meow Meow")

boy = Animal.Boy(3, "Adam", "Grabarz")

boy.print\_information() # 3, Adam, I'm mammal - klasa Boy

dziedziczy print\_information z klasy Mammal

boy.give\_voice() # Little boy wants to play.

boy.go\_to() # go go go

animal = Animal.Animal(5, "Ciapek")

animal.print\_information() # 5, Ciapek

animal.give\_voice() # def give\_voice(self): raise NotImplementedError()

# wywołanie tej funkcji spowoduje wyświetlenie naszego zdefiniwanego błędu:

#Traceback (most recent call last):

# File "d:/python/basics/strefakursow/klasy/05 dziedziczenie.py", line 15, in <module>

#   animal.give\_voice() # def give\_voice(self): raise NotImplementedError()

# File "d:\python\basics\strefakursow\klasy\klasa05dziedziczenie.py", line 9, in give\_voice

#   raise NotImplementedError() # definiujemy obsługę błędu, jeśli odniesiemy się do tej funkcji, to ją wywołamy

#NotImplementedError

class Animal:

    def \_\_init\_\_(self, age, name):

        self.name = name

        self.age = age

    def print\_information(self):

        print(self.age)

        print(self. name)

    def give\_voice(self):

        raise NotImplementedError() # definiujemy obsługę błędu, jeśli odniesiemy się do tej funkcji, to ją wywołamy

class Mammal(Animal): # mammal - ssak

    def \_\_init\_\_(self, age,name):

        Animal.\_\_init\_\_(self, age, name)

    def go\_to(self):

        print("go go go")

    def print\_information(self):

        Animal.print\_information(self) #klasa Mammal dziedziczy

print\_information z klasy Animal

        print("I'm mammal")

class Boy(Mammal):

    def \_\_init\_\_(self, age, name, surname):

        Mammal.\_\_init\_\_(self, age, name)

        self.surname = surname

    def give\_voice(self):

        print("Little boy wants to play.")

    def print\_information(self):

        Mammal.print\_information(self) # klasa Boy dziedziczy print\_information z klasy Mammal(a klasa Mammal z klasy Animal)

        print(self.surname)

        print("i'm a boy")

**klasy(metody magiczne)**

class punkt2D:

    def \_\_init\_\_(self, x , y):

        self.x = x

        self.y = y

        self.odleglosc = math.sqrt(x\*\*2 + y\*\*2) # liczymy przeciwprostokątną trójkąta prostokątnego

                                                # a2+b2=c2 -> c=pierwiastek(a2+b2). (X , Y) jest to punkt w układzie

                                                # współrzędnych wiec c będzie to odległość do początku układu współrzędnych

def \_\_add\_\_(self, drugi):

        return punkt2D(self.x + drugi.x, self.y + drugi.y)

# definiujemy tak, aby policzyć p3 = p1 + p2

    def \_\_lt\_\_(self, drugi):

        return self.odleglosc < drugi.odleglosc

    def \_\_le\_\_(self, drugi):

        return self.odleglosc <= drugi.odleglosc

    def \_\_eq\_\_(self, drugi):

        return self.x == drugi.x and self.y == drugi.y

    def \_\_len\_\_(self): # \_\_len\_\_ - długość - w tym przypadku od

# początku układu współrz.

        return int(round(self.odleglosc, 0))

    def \_\_str\_\_(self):

        return f"współrzędna x: {self.x}, współrzędna y: {self.y}"

    def \_\_repr\_\_(self):

        text = repr(punkt2D) # wyświetli typ klasy

        idobj = id(punkt2D) # wyświetli id naszego obiektu

        return f"typ: {text}\nid obiektu: {idobj}\nwspółrzędne punktu: {self.x}, {self.y}"

p1 = punkt2D(2, 5)

p2 = punkt2D(4, 5)

p3 = p1 + p2 # możliwe dzięki def \_\_add\_\_(self, drugi):

print(p3.x) # 6

print(p3.y) # 10

print(p1.odleglosc) # 5.385164807134504

print(p2.odleglosc) # 6.4031242374328485

print(p1 < p2) # True, możliwe dzięki def \_\_lt\_\_(self, drugi):

print(p1 == p2) # False,możliwe dzięki def \_\_eq\_\_(self, drugi):

print(p2 == p2) # True,możliwe dzięki def \_\_eq\_\_(self, drugi):

print(len(p3)) # 12, możliwe dzięki def \_\_len\_\_(self):

print(p3.odleglosc) # 11.661903789690601

print(p1) # współrzędna x: 2, współrzędna y: 5 , możliwe dzięki def \_\_str\_\_(self):

print(repr(p1)) # typ: <class '\_\_main\_\_.punkt2D'>

                # id obiektu: 2977001692512

                # współrzędne punktu: 2, 5

# MAGICZNE METODY

# \_\_init\_\_ - # specjalna metoda inicjalizacji danych (konstruktor). Obiekt powstały w oparciu klasę z

 konstruktorem \_\_init\_\_ przejmuje jego parametry (bez self)

# \_\_add\_\_ - dodawanie

# \_\_lt\_\_ - less then - mniejsze od

# \_\_le\_\_ - less equal - mniejsze rowne od

# \_\_eq\_\_ - equal - równe

# \_\_gt\_\_ - grader then - większe od

# \_\_ge\_\_ - grader equal - wieksze rowne od

# \_\_ne\_\_ - not equal - różne od

# \_\_eq\_\_ - equal - równe

# \_\_len\_\_ - długość

# \_\_del\_\_ - delete - metoda wykonywana jest na sam koniec skryptu i służy wyczyszczeniu pamięci.

# \_\_new\_\_ - (przeznaczony dla zaawansowanych programistów)

# \_\_str\_\_ - Metoda \_\_str\_\_ odpowiedzialna jest za zwrócenie tekstu zdefiniowanego podczas tworzenia klasy.

# \_\_repr\_\_ zwraca reprezentację danego obiektu w postaci string

# \_\_call\_\_ - metoda powoduje, że obiekt, ktory utworzymy będzie można wywołać jak funkcję

**Dekoratory (statyczne, klasowe)**

from klasa10dekoratory\_statyczneklasowe import Numbers

from klasa101dekoratory\_statyczneklasowe import Animal

przykład = Numbers() # tworzymy obiekt (klasa Numbers)

przykład.addNumber(3)  # korzystając z def add.number() dodajemy elementy do naszej listy self.numbers = numbers

przykład.addNumber(5) przykład.addNumber(8)

przykład.addNumber(6) przykład.addNumber(13)

przykład.addNumber(7) przykład.addNumber(2)

print(przykład.sumNumbers()) # 44 - z def z klasy Numbers

print(przykład.multiNumbers()) # 131040

print(Numbers.subtractNumbers(15, 5)) # 10 metoda statyczna korzystamy z klasy pomijając tworzenie instancji(obiektu)

Numbers.printInformation() # metoda klasyczna możę pobierać dane z naszej klasy nasz printInformation() ma cls.my\_numbers +=1 więc dodaje 1 do pola my\_numbers po wywoł. Metody klasycznj printInformation() każde odniesienie do my\_numbers będzie +1

print("Pole w klasie:") print("po @classmethod:")

print(przykład.my\_numbers) # 10 print(przykład.my\_numbers) # 11

print(Numbers.my\_numbers) # 10 print(Numbers.my\_numbers) # 11

kot1 = Animal(Animal,"Łapek") # Tworzymy obiekt. Ponieważ metoda klasowa ma wbudowany inicjalizator( a nie klasa głowna)

kot2 = Animal(Animal,"Łapek") # tworząc obiekt na klasie Animal, musimy dla naszej  @classmethod podać rownież, że korzysta ona z klasy Animal

kot3 = Animal(Animal,"Łapek")

kot4 = Animal(Animal,"Łapek")

kot5 = Animal(Animal,"Łapek")

print(kot5.howManyAnimal) # metoda cls.howManyAnimal po każdym utworzonym obiekcie będzie zwiększała o 1 nasze pole(zmienną) howManyAnimal znajdującą się bezpośrednio w naszej klasie

class Numbers:

    my\_numbers = 10

    def \_\_init\_\_(self, numbers = []):

        self.numbers = numbers

    def sumNumbers(self): # funkcja ma zsumować wszystkie

        elementy listy self.numbers

summary = 0

        for x in self.numbers:

            summary += x

        return summary

    def multiNumbers(self): # przemnoży przez siebie wszystkie

        elementy listy self.numbers

product = 1

        for x in self.numbers:

            product \*= x

        return product

    def addNumber(self, number): # funkcja pozwoli dodać

        argumenty do listy self.numbers

self.numbers.append(number)

    @staticmethod # tworzymy funkcję(metodę) statycznie.

Możemy z niej korzystać bez tworzenia instancji(obiektu)

    def subtractNumbers(a,b):

        return a - b

    @classmethod # metoda klasowa może uzyskać dane z naszej

klasy (w odróżnieniu od @staticmethod )

    def printInformation(cls): # (cls) - musi być w metodzie,

żeby mogła uzyskać dane z naszej klasy

        print("Jestem class method")

        print ("na rzecz klasy", cls.\_\_name\_\_) # cls.\_\_name\_\_ -

wyświetli nazwę naszej klasy

        cls.my\_numbers +=1 # w metodzie klasycznej możemy odnosić się do pól z naszej klasy np. my\_numbers i je zmieniać

class Animal:

    howManyAnimal = 0

    @classmethod

    def \_\_init\_\_(cls, self, name\_animal): # metoda klasowa z

wbudowanym inicjalizatorem

        self.name\_animal = name\_animal

        cls.howManyAnimal += 1

**Klasy (live cycle, destruktory)**

class test:

    def \_\_del\_\_(sel): # metoda wykonywana jest na sam koniec skryptu i służy wyczyszczeniu pamięci.

                      # Nawet jeśli wywołamy obiekt, który powstał na bazie klasy z tą magiczną metodą,

                      # a po nim pojawi się dalsza część kodu, to i tak obiekt zostanie wywołany na końcu skryptu

        print("Bye class")

# /1/

obj = test()

print("koniec")

del obj

# konsola wyświetli: koniec

#                    Bye class

# /2/

obj = test()

del obj         # usuwamy obiekt obj przed wywołaniem print("koniec"), więc nie odwróci on kolejności

print("koniec") # wyświetlenia (wykonania) jak dla przypadku wyżej /1/

# konsola wyświetli: Bye class

#                    koniec

# /3/

obj = test()

obj2 = obj     # tworzymy kolejny obiekt obj2, który powstaje na bazie obiektu obj (tworzymy uchwyt do klasy),

del obj        # więc wywołanie funkcji del nie spowoduje calkowitego usunięcia obiektu obj (uchwyt pozostanie)

print("koniec")

del obj2

# konsola wyświetli: koniec

#                    Bye class

# /4/

obj = test()

obj2 = obj

lista = [obj2] # sytuacja jak /3/, utworzona lista ma uchwyt do klasy (obj, obj2 są nie calkowicie usunięte)

del obj

del obj2

print("koniec")

del lista[0]

# konsola wyświetli: koniec

#                    Bye class

# /5/

obj = test()

obj2 = obj

lista = [obj2]

del obj

del obj2

del lista[0]

print("koniec")

# konsola wyświetli: Bye class

#                    koniec

# Zasada usuwania obiektów jest taka, że jśli jakiś inny obiekt odnosi się do kasowanego funkcją del obiektu,

# to ten obiekt nie zostanie skasowany całkowice i pozostawi uchwyt do klasy.

# Całkowite kasowanie obiektu jest możliwe tylko wtedy, gdy nie ma on relacji z żadnym innym obiektem

**Klasy hermetyzacja (urywanie danych)**

class Test:

    \_lista = []                     # \_ podkreślenie oznacza, że lista jest prywatna (jest to tylko informacja,

    def dodaj(self, arg):           # (dalej możemy się do niej odnieść poza klasą)

        self.\_lista.append(arg)

    def zdejmij(self):

        if len(self.\_lista) > 0:

            return self.\_lista.pop(len(self.\_lista) - 1) # usuwa ostatni argument listy (pop)

        else:

            return

obj = Test() obj.dodaj("A") obj.dodaj("B") obj.dodaj("C")

obj.\_lista.append("X") # zastosowanie \_ podkreślenia powoduje, że możemy (w tym przypadku) dodać "X" do listy

print(obj.\_lista) # ['A', 'B', 'C', 'X']

print(obj.zdejmij()) # X

print(obj.\_lista) # ['A', 'B', 'C']

class Test1:

    \_\_lista = []                     # \_\_podkreślenie oznacza, że lista jest prywatna (jest to tylko informacja,

    def dodaj(self, arg):           # (dalej możemy się do niej odnieść poza klasą, jednak musimy wiedzieć jak)

        self.\_\_lista.append(arg)

    def zdejmij(self):

        if len(self.\_\_lista) > 0:

            return self.\_\_lista.pop(len(self.\_\_lista) - 1) # usuwa ostatni argument listy (pop)

        else:

            return

obj = Test1() obj.dodaj("A") obj.dodaj("B") obj.dodaj("C")

obj.\_Test1\_\_lista.append("X") # zastosowanie odwołania do klasy \_NAZWAKLASY (w naszym przypadku \_Test1) daje nam

                              # możliwość odniesienia się do naszej listy (utworzonej w klasie) poza klasą

# obj.\_\_lista.append("X") # zapis nieprawidłowy, wywoła błąd

print(obj.zdejmij()) # X

print(obj.\_Test1\_\_lista) # ['A', 'B', 'C']

# print(obj.\_\_lista) # zapis nieprawidłowy, wywoła błąd

# nie zastosowanie \_Test1 w metodzie obj.\_\_lista.append("X") spowoduje wywołanie błędu. W jakimś stopniu

# zabezpiecza nam to naszą listę przed ingerencją w nią z poza klasy (trzeba wiedzieć czego użyć aby to zrobić)

# Daje to nam namiastkę hermetyzacji (ukrywania danych) w klasie

**Klasy (gettery, settery)**

# property – własności # getter, setter - ustawiacz, pobieracz

class KontoBankowe:

    \_\_stan = 0

    @property               # tworzenie właściwości. definicja przestaje pełnić rolę funkcji

    def stan\_konta(self):   # a jest traktowana prawie jak zwykła zmienna (ale jest tylko do odczytu)

        return self.\_\_stan  # jeśli byśmy spróbowali ją wywołać i nadać jej inną wartość (konto.stan\_konta = 50) - błąd

                            # będzie to możliwe dopiero po użyciu gettera i settera

    @stan\_konta.getter      # tworzenie gettera dla @property - da nam to możliwość ustawienia nowego np. zapisu we właściwości

    def stan\_konta(self):

        return "stan konta: " + str(self.\_\_stan) + "zł"

    @stan\_konta.setter      # tworzenie settera dla @property - da nam to możliwość w trakcie wywołania pobrać inną wartość

    def stan\_konta(self, value): # w tym przypadku (self.\_\_stan += value) zwiększymy stan naszego konta o argument value

        self.\_\_stan += value

konto = KontoBankowe()

print(konto.stan\_konta) # stan konta: 0zł - korzystając z @property def stan\_konta nie użuwamy (),

# bez @property - print(konto.stan\_konta()) - jakbyśmy próbowali to tak wywołać, spowoduje to błąd

konto.stan\_konta = 50 # stan konta: 50zł (+50)

print(konto.stan\_konta)

konto.stan\_konta = 100 # stan konta: 150zł (+100)

print(konto.stan\_konta)

konto.stan\_konta = -125 # stan konta: 25zł (-125)

print(konto.stan\_konta)

pola (gettery, settery)

obiekt = Example()

print(obiekt.get\_example\_field()) # 10

obiekt.set\_example\_field(50)

print(obiekt.get\_example\_field()) # 50

obiekt.example\_field = -40 # if value\_field < 0:

self.\_\_example\_field = 0

print(obiekt.get\_example\_field()) # 0

obiekt.set\_example\_field(-6050) #

print(obiekt.get\_example\_field()) # 0

class Example:

    example\_field = 10

    def \_\_init\_\_(self):

        self.\_\_example\_field = 10

    def get\_example\_field(self):

        return self.\_\_example\_field

    def set\_example\_field(self, value\_field):

        if value\_field < 0:

            self.\_\_example\_field = 0

        else:

            self.\_\_example\_field = value\_field

    example\_field = property(get\_example\_field, set\_example\_field)

**Klasy obiekt wywoływany \_\_call\_\_(jako funkcja)**

liczba = Func(4)

print(liczba(10)) # 40 - wywołujemy obiekt jak funkcję

(\_\_call\_\_). W tym przypadku times = 10

napis = Func("\_Michał Grabarz\_")

print(napis(6)) # \_Michał Grabarz\_\_Michał Grabarz\_\_Michał Grabarz\_\_Michał Grabarz\_\_Michał Grabarz\_\_Michał Grabarz\_

class Func:

    def \_\_init\_\_(self, value):

        self.value = value

    def \_\_call\_\_(self, times): # metoda powoduje, że obiekt,

ktory utworzymy możemy wywołać jak funkcję

        if type(self.value) == int:

            return self.value \* times

        elif type(self.value) == str:

            return self.value \* times

**TKinter**

import tkinter as tk

from tkinter import ttk # ttk - klasa do tworzenia label

win = tk.Tk() # tworzymy obiekt

win.title("Pierwsze okno w tkinter") #odwołujemy się do naszego obiektu i nazywamy go

# win.resizable(0,0) # nię będziemy mogli rozszerzyć okna (0,0)

# label

aLabel = ttk.Label(win, text = "Pierwszy Label") # tworzymy obiekt który będzie  label

aLabel.grid(column = 0, row = 0) # umieszczamy go w oknie

# Button Click Event Function

def clickMe(): # funkcja z różnymi zachowaniami naszych obiektów

    action.configure(text = "\*\* Zostałem kliknięty! \*\*") # zmiana tekstu po kliknięciu

    aLabel.configure(text = "Drugi Label") # po kliknięciu zmieni teks w naszym aLabel

    aLabel.configure(foreground = "red") # nowy tekst będzie miał kolor czerwony

# Dodanie Button

action = ttk.Button(win, text = "Kliknij mnie", command = clickMe) # tworzymy przycisk, command- odnosi się do funkcji ze zdefiniowanymi zachowaniami po kliknięciu button

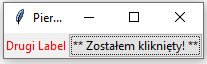
action.grid(column = 1, row = 0) # umieszczamy go w oknie

win.mainloop() # okno cały czas będzie wyświetlane

Przed kliknięciem:



**Po kliknięciu:**

****

import tkinter as tk

from tkinter import ttk # ttk - klasa do tworzenia label

from tkinter import scrolledtext # klasa do tworzenia scrolledtext

win = tk.Tk() # tworzymy obiekt

win.title("Pierwsze okno w tkinter")

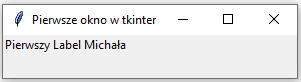
aLabel = ttk.Label(win, text= "Pierwszy Label Michała")

aLabel.grid(column= 0, row= 0)

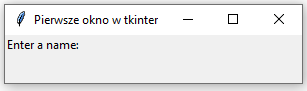
# Zmieniamy nasz Label

aLabel.configure(text = "Enter a name: ") # nadpisujemy nasz aLabel

# ttk.Label(win, text = "Enter a name: ").grid(column = 0, row = 0) # gdybyśmy zostawili nowy przycisk w tym samym miejscu, przesłoniłby on stary aLabel



Po nadpisaniu aLabel

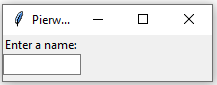


# Dodanie Textbox

name = tk.StringVar() # zmienna będzie przechowywała nasz wpisany w Textbox tekst

nameEntered = ttk.Entry(win, width= 12, textvariable= name) # tworzymy nowy obiekt o długości = 12(width = 12, pole do wpisania tekstu), będzie się on odnosił do naszej zmiennej name(textvariable = name)

nameEntered.grid(column= 0, row= 1) # umieszczamy go w oknie



# Modyfikacja funkcji klik button

def clickMe():

    action.configure(text= "Hello " + name.get())

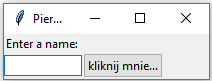
# Dodanie Button (przycisku)

action = ttk.Button(win, text= "kliknij mnie...", command= clickMe)

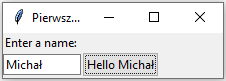
action.grid(column= 2, row= 1)

# action.configure(state = "disabled") # nasz obiekt(Button) będzie nieaktywny

nameEntered.focus() # spowoduje podświetlenie Textbox do wpisania tekst



Po kliknięciu:



# Dodanie Combobox(lista wartości rozwijana )

def clickMe(): # jeszcze raz modyfikujemy naszą funkcję cilckMe

    action.configure(text= "Hello " + name.get() + " " + numberChosen.get())

ttk.Label(win, text = "Wybierz numer: ").grid(column= 1, row= 0)

number = tk.StringVar # tworzymy zmienną, która przechowa nasz wprowadzony numer

numberChosen = ttk.Combobox(win, width= 12, textvariable= number) # tworzymy zmienną z

naszym Combobox

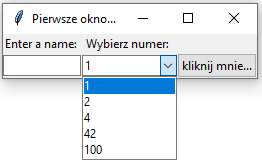
# numberChosen = ttk.Combobox(win, width= 12, textvariable= number, state= "readonly")- dodanie zmiennej state= "readonly" zablokuje nam wpisywanie w naszym Combobox danych z klawiatury (będziemy mogli tylko wybrać z listy)

numberChosen["values"] = (1, 2, 4, 42, 100) # definiujemy naszą rozwijaną listę (w

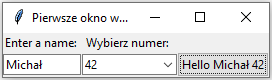
postaci krotki)

numberChosen.grid(column= 1, row= 1)

numberChosen.current(0) # określamy, który element krotki będzie pokazany jako pierwszy



Po akcji:



# Dodanie przycisków Checkbox (nieaktywny zaznaczony)

chVarDis = tk.IntVar() # obiekt do przechowywania wartości z Checkbox

check1 = tk.Checkbutton(win, text= "disabled", variable=chVarDis, state= "disabled")

# tworzymy Checkbox 1, state= disabled - nieaktywny

check1.select() # .select - będzie zaznaczony

check1.grid(column= 0, row= 4, sticky= tk.W) # sticky= tk.W - wyrównywanie do granicy

West (E- East)

# Dodanie przycisków Checkbox (aktywny niezaznaczony)

chVarUn = tk.IntVar()

check2 = tk.Checkbutton(win, text= "UnChecked", variable=chVarUn)

check2.deselect() # .deselect - nie będzie zaznaczony

check2.grid(column= 1, row= 4, sticky= tk.W)

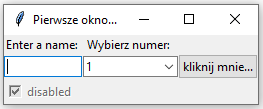
# Dodanie przycisków Checkbox (aktywny zaznaczony)

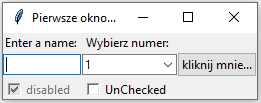
chVarEn = tk.IntVar()

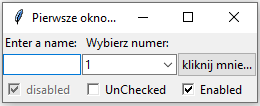
check3 = tk.Checkbutton(win, text= "Enabled", variable=chVarEn)

check3.select()

check3.grid(column= 2, row= 4, sticky= tk.W)







# RadioButton

Color1 = "Blue" # definiujemy kolory

Color2 = "Gold"

Color3 = "Red"

def radCall(): # tworzymy funkcję przypisującą kolor tła w zależności od naszego wyboru

    radSel = radVar.get() # tworzymy zmienną, która weźmie dane z naszej zmiennej

# przechowującej dane

    if radSel == 1: win.configure(background=Color1)

    elif radSel == 2: win.configure(background=Color2)

    elif radSel == 3: win.configure(background=Color3)

radVar = tk.IntVar() # zmienna przechowująca dane z naszego RadioButton

rad1 = tk.Radiobutton(win, text=Color1, variable=radVar, value=1, command=radCall) # value=1 - radiobutton zwróci taką wartość, kiedy zostanie wciśnięty

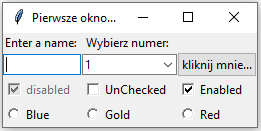
rad1.grid(column=0, row=5, sticky=tk.W, columnspan=3)

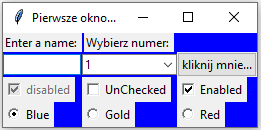
rad2 = tk.Radiobutton(win, text=Color2, variable=radVar, value=2, command=radCall)

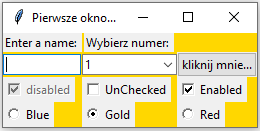
rad2.grid(column=1, row=5, sticky=tk.W, columnspan=3)

rad3 = tk.Radiobutton(win, text=Color3, variable=radVar, value=3, command=radCall)

rad3.grid(column=2, row=5, sticky=tk.W, columnspan=3)







# RadioButton w pętli

colors = ["Green", "Black", "White"]

def radCall1():

    radSel1 = radVar1.get()

    if radSel1 == 0: win.configure(background=colors[0])

    elif radSel1 == 1: win.configure(background=colors[1])

    elif radSel1 == 2: win.configure(background=colors[2])

    # for c in range(3): # można też użyć zapisu z pętlą for (przy dużej liczbie opcji)

    #     if radSel1 == c: win.configure(background=colors[c])

radVar1 = tk.IntVar()

radVar1.set(99) # definiujemy radVar1 tak, aby nie była ona równa 0 lub 1 lub 2 (nie

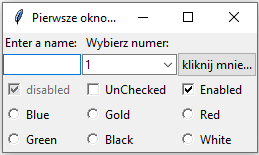
# wyświetliła radsel1 w chwili uruchomienia skryptu)

for col in range(3):

    curRad = "rad" + str(col) #pętla przypisze nazwę curRad w zależności, jaki będzie col

    curRad = tk.Radiobutton(win, text=colors[col], variable= radVar1, value=col, command=radCall1)

    curRad.grid(column=col, row=6, sticky=tk.W)



# Scrolled Text Control - pole (podobne do Textbox) o ustalonej wysokości i szerokości

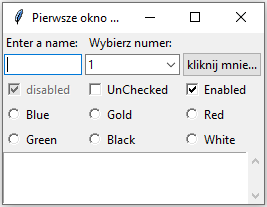
scrolW = 30

scrolH = 3

scr = scrolledtext.ScrolledText(win, width=scrolW, height=scrolH, wrap= tk.WORD)# width- szerokość pola, high- wysokość pola, wrap=tk.WORD - dzielenie i zawijanie co całe słowa

scr.grid(column=0, columnspan=3) # columnspan = 3 - umieści nasze okno w naszych 3 już

# wykorzystywanych kolumnach



# Label Frame (Label otoczony ramką, w której mogą występować inne elementy)

labelsFrame = ttk.LabelFrame(win, text= "--- Labels in Frame ---") # tworzymy label Frame

# labelsFrame.grid(column=0, row=8, padx= 10, pady= 10)# padx- ustawienie do lewej i # prawej, pady- ustawienie do góry i do dołu (odnosi się w tym przypadku do naszego okna # win. W pikselach)

labelsFrame.grid(column=0, row=8, columnspan= 3, sticky= "WE")# sticky= "WE" – wyrównanie

# do lewej i prawej (West, East)

ttk.Label(labelsFrame, text= "Label1").grid(column=0, row=0) # tworzymy w nim labele

# (numeracja kolumn i rzędów od 0)

ttk.Label(labelsFrame, text= "Label2").grid(column=0, row=1)

ttk.Label(labelsFrame, text= "Label3").grid(column=0, row=2)

for child in labelsFrame.winfo\_children(): # pętla pozwoli ustawić padding na

# poszczególne labels w LabelsFrame

    child.grid\_configure(padx=10, pady=10) # ustawi padding dla każdego pojedyńczego

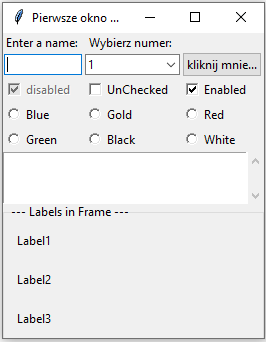
# label

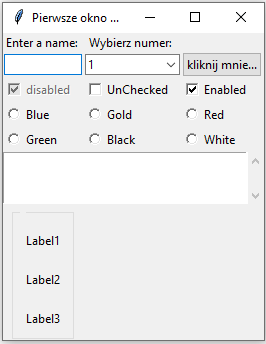
# metoda child - pozwala odnieść się do naszych zdefiniowanych elementów labels w naszym LabelsFrame

# winfo\_children() - pozyskuje informacje o położeniu naszych labels względem LabelFrame

# grid\_configure() - pozwala skonfigurować nowe położenie dla poszczególnego elementu

# usunięcie text z LabelFrame oraz columnspan i sticky z labelsFrame.grid pozwoli zobaczyć jak działa padding





**# Menu**

from tkinter import Menu

from tkinter import messagebox as mBox # moduł do tworzenia  komunikatów dla użytkownika

def \_quit(): # akcja odpowiedzialna za zamknięcie naszego programu

    win.quit()

    win.destroy()

    exit()

menuBar = Menu(win) # tworzymy obiekt przypięty do okna win

win.config(menu=menuBar) # configurujemy, aby nasze okno odnosiło się do naszego MemuBar

# dodajemy elementy do naszego menu (add menu items)

fileMenu = Menu(menuBar, tearoff=0) # tworzymy obiekt menu w naszym menuBar. tearoff-

# wyłącza przesuwanie menu względem okna

fileMenu.add\_command(label="New") # dodajemy w naszym fileMenu

fileMenu.add\_command(label="Save")

fileMenu.add\_separator() # tworzy separator pomiędzy etykietami

fileMenu.add\_command(label="Exit", command=\_quit) # korzystamy z def \_quit

menuBar.add\_cascade(label="File", menu=fileMenu) # głownę pole w naszym menuBar(z niego

# rozwinie się możliwość kliknięcia "New"). File-->New

def \_msgBoxAbout(): # tworzymy funkcję do wyświetlenia komunikatu dla użytkownika

    mBox.showinfo("Wiadomość", "Strefa kusrów - kurs Python") # showinfo("nazwa

komunikatu", "treść komunikatu")- metoda wyświetli komunikat (po akcji)

def \_msgBoxWarning(): # tworzymy funkcję do wyświetlenia komunikatu dla użytkownika

mBox.showwarning("Ostrzeżenie", "Uwaga - Nowe kursy w strefie kursów. Zapraszam") #

showwarning("nazwa komunikatu", "treść komunikatu")- metoda wyświetli użytkownikowi ostrzeżenie (po wykonaniu akcji)

def \_msgBoxError(): # tworzymy funkcję do wyświetlenia komunikatu dla użytkownika

    mBox.showerror("Uwaga", "Aplikacja może nie działać prawidowo") # showerror("nazwa

komunikatu", "treść komunikatu")- metoda wyświetli error (po akcji)

def \_msgBoxAskYesNo():

    answer = mBox.askyesno("Komunikat", "Czy jesteś zadowolony z kursu?") # przypisujemy

zmienną do naszego pytania tak/nie

    print(answer) # jeśli tak, zwróci do konsoli True, nie - False

helpMenu = Menu(menuBar, tearoff=0)

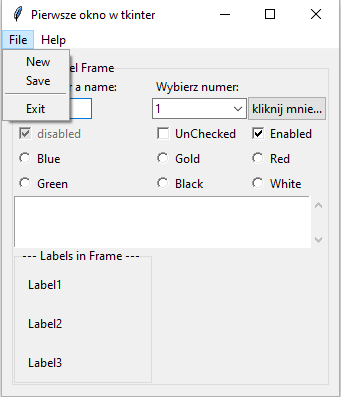
helpMenu.add\_command(label="About", command=\_msgBoxAbout) # podpinamy akcję  \_msgBox do labela "About"

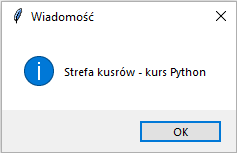
helpMenu.add\_command(label="Warning", command=\_msgBoxWarning)

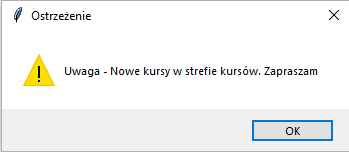
helpMenu.add\_command(label="Error", command=\_msgBoxError)

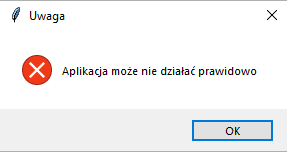
helpMenu.add\_command(label="Question", command=\_msgBoxAskYesNo)

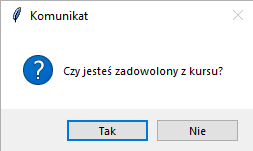
menuBar.add\_cascade(label="Help", menu=helpMenu)











**# spinbox** (pole zawierające warości integer ze strzałkami góra/dół do ich ustawienia)

from tkinter import Spinbox

def \_spin(): # Definiujemy akcję dla naszego spina

    value = spin.get() # pobieramy wartośc z naszego spin

    print(value) # wydruk do terminala naszej wartości spin

    scr.insert(tk.INSERT, value + " ") # umieszczamy nasz value w naszym scr(scrolled

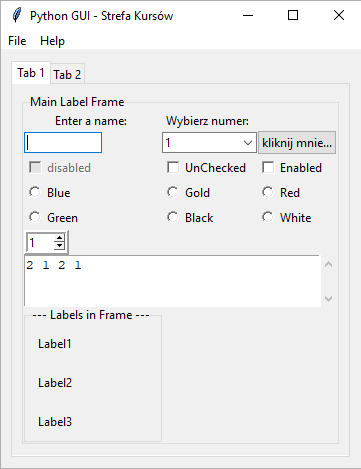
# Text) z odpowiednim formatowaniem(arg 2)

spin = Spinbox(mainFrame, from\_=1, to=10, width=4, bd=4, command=\_spin, relief='ridge') #

from\_=1, to=10 - zakres wyświetlanych int, width=4 szerokość w pikselach, bd=4 ramka w pikselach(bd od border)

spin.grid(column=0, row=7, sticky=tk.W) # relief="ridge" - formatowanie ramki (flat, groove, raised, ridge, solid, or sunken). Dopuszcza się zapis relief=tk.RIDGE"

# spin["values"] = (10,20,30,40,50) # możemy nadpisać wartości w naszym spinbox



**# Tab Control (zakładki)**

tabControl = ttk.Notebook(win) # tworzymy obiekt na klasie(metoddzie NoteBook)

tab1 = ttk.Frame(tabControl) # tworzymy 1 zakładkę i podpinamy do tabControl

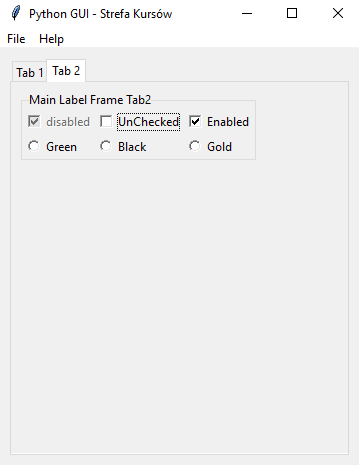
tabControl.add(tab1, text="Tab 1") # dodajemy do naszego tabControl naszą zakładkę tab1 i definiujemy jej nazwę text="Tab 1"

tab2 = ttk.Frame(tabControl) # tworzymy 2 zakładkę

tabControl.add(tab2, text="Tab 2")

tabControl.pack(expand=1, fill="both", padx=10, pady=10) # fill="both" powoduje rozciągnięcie naszego tabControl do szerokości i wysokości naszego okna win (może być jeszcze fill="x" - rozciągnie tylko w poziomie, fill="y" - rozciągnie tylko w pionie)

# ja zastosowałem padx, pady aby część okna win była widoczna ( i działały nasze RadioButton z kolorami okna win)



**# messagebox**

from tkinter import messagebox as mBox

from tkinter import Tk

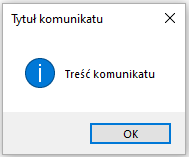
root = Tk() # tworzymy okno

root.withdraw()  # spowoduje wyświetlenie naszego mBox i nie wyświetlenie okna

mBox.showinfo("Tytuł komunikatu", "Treść komunikatu")

root.mainloop()

# może być przydatne w przypadku błedu program



win.iconbitmap('./ikona2.ico') # pokaże naszą ikonę w oknie win

**# ToolTip (komentarze, podpowiedzi do widgetów)** – patrz plik /przydatne/ToolTip komentarze.py

**Label**

* **activebackground** kolor tła, gdy kontrolka jest aktywna
* label = tk.Label(window, activebackground = ***"#ff0000"***)
* **activeforeground** kolor pierwszoplanowy, gdy kontrolka jest aktywna
* **anchor** kontroluje, gdzie tekst kontrolki (lub obrazu) powinien być usytuowany. Użyj jako ustawienie **N** (north), **NE** (north-east), **E** (east), **SE** (side-east), **S** (side), **SW** (side-west), **W** (west), **NW** (north-west) lub **CENTER**;
* **background** kolor tła
* **bg** to samo co background
* **bitmap** bitmapa wyświetlana w kontrolce, jeżeli opcja **image** jest ustawiona ta pozycja jest ignorowana;
* **borderwidth** szerokość obramowania kontrolki;
* **bd** to samo co **borderwidth**
* **compound** kontroluje jak tekst i grafika są wyświetlane w kontrolce. Przy domyślnych ustawieniach, gdy opcja **image** lub **bitmap** są ustawione, wtedy grafika jest rysowana zamiast tekstu. Gdy opcja ta jest ustawiona na **CENTER** wtedy tekst jest rysowany nad grafiką. Gdy opcja ta jest ustawiona na **BOTTOM**, **LEFT**, **RIGHT** lub **TOP** tekst jest wyświetlany odpowiednio pod, po lewej, po prawej lub ponad grafiką. Domyślna wartość to **NONE**.
* **cursor** określa jaki kursor będzie wyświetlany, gdy ten znajdzie się w obszarze kontrolki.
* **disabledforeground** jaki kolor pierwszoplanowy ustawić, gdy kontrolka jest nieaktywna.
* **font** czcionka, jakiej użyć do wyświetlania tekstu, przykład:

Listing 4ZwińKopiuj

* label = tk.Label(window, font = (***"Times New Roman"***, 20)) # krój czcionki, rozmiar czcionki
* **foreground** kolor czcionki
* **fg** to samo co **foreground**
* **height** wysokość kontrolki, gdy kontrolka wyświetla tekst używane są jednostki tekstu, natomiast w przypadku grafiki piksele. Gdy ten parametr jest ustawiony na zero to wysokość jest obliczana na podstawie zawartości.
* **highlightbackground** kolor podświetlenia ramki, gdy kontrolka nie ma fokusa;
* **highlightcolor** kolor podświetlenia ramki, gdy kontrolka ma fokus;
* **highlightthickness** szerokość ramki podświetlenia. Domyślna wartość ustawiona jest na 0;
* **image** obrazek do wyświetlenia, domyślnie wartość powinna być obiektem klasy PhotoImage lub ButmapImage lub odpowiadający im obiekt;
* **justify** wyrównanie tekstu: **LEFT**, **RIGHT** lub **CENTER**;
* **padx** dodatkowe przesunięcie w poziomie tekstu, domyślna wartość to 1;
* **pady** dodatkowe przesunięcie w pionie tekstu. Domyślna wartość to 1;
* **relief** rodzaj dekoracji obramowania. Ustawienie domyślne to **FLAT**, inne dostępne to **SUNKEN**, **RAISED**, **GROOVE** i **RIDGE**;
* **state** definiuje sposób renderowania kontrolki. Domyślna ustawienie to **NORMAL**, inne dostępne wartości to **ACTIVE** i **DISABLED**
* **takefocus** jeżeli ustawione na **true** to kontrolka akceptuje fokus, domyślnie ustawione na **false**;
* **text** tekst do wyświetlenia;
* **textvariable** połączone z zmiennymi **tkinter** (zazwyczaj StringVar). Gdy zmienna ulega zmianie, tekst w kontrolce się aktualizuje. Przykład:
* **underline** podkreślenie tekstu wyświetlanego, domyślnie wyłączone i ustawione na -1;
* **width** szerokość kontrolki. Gdy wyświetlany jest tekst wtedy używane są jednostki związane z tekstem. W przypadku grafiki jednostkami są piksele. Domyślnie ustawione na zero co oznacza, że szerokość będzie obliczana na podstawie zawartości kontrolki.

**Entry (TextBox)**

Konstruktor klasy **Entry** przyjmuje taką samą listę argumentów co klasy **Label**, jedynie opcje nieco się różnią:

* **background** - kolor tła kontrolki;
* **bg** - to samo co **background**;
* **borderwidth** - szerokość obramowania;
* **bd** - to samo co **borderwidth**;
* **cursor** - określa typ kursora wyświetlanego, gdy myszka znajduje się nad obszarem kontrolki;
* **disabledbackground** - tło, które będzie użyte gdy kontrolka jest nieaktywna.
* **disabledforeground** - kolor tekstu, gdy kontrolka jest nieaktywna;
* **exportselection** - gdy ustawione jest na **True** to zaznaczony tekst automatycznie jest kopiowany do systemowego schowka. Domyślnie wartość ta jest ustawiona na **True**;
* **font** - czcionka wyświetlanego tekstu;
* **foreground** - kolor tekstu;
* **fg** - to samo co **foreground**
* **highlightbackground** - razem z **highlightBackground** opcja ta kontroluje jak rysować podświetlenie ramki kontrolki, gdy nie ma fokusa.
* **highlightcolor** - to samo co **highlightbackground**, z tą różnicą, że używa się tej opcji przy rysowaniu podświetlenia obramowania gdy kontrolka ma fokus;
* **highlightthickness** - szerokość ramki podświetlania kontrolki;
* **insertbackground** - kolor karetki wpisywania tekstu;
* **insertborderwidth** - długość obramowania karetki wpisywania tekstu. Gdy ten parametr jest ustawiony na zero, używany jest styl **RAISED**;
* **insertofftime** - razem z opcją **insertontime** kontroluje prędkość mrugania karetki wpisywania tekstu. Obie wartości są wyrażone w milisekundach;
* **insertontime** - podobnie jak **insertofftime**, tylko że określa czas pojawiania się karetki kursora wpisywania tekstu;
* **insertwidth** - długość karetki wpisywania tekstu wyrażona w pikselach.
* **justify** - określa w jaki sposób tekst jest pozycjonowany wewnątrz kontrolki. Dostępne są: **LEFT**, **CENTER** lub **RIGHT**. Domyślna wartość to **LEFT**;
* **readonlybackground** - kolor tła kontrolki, gdy ta jest w trybie "readonly" (tylko do odczytu);
* **relief** - styl obramowania kontrolki. Domyślna wartość to **SUNKEN**, inne możliwe to: **FLAT**, **RAISED**, **GROOVE** i **RIDGE**;
* **selectbackground** - kolor tła zaznazcenia;
* **selectborderwidth** - szerokość obramowania zaznaczenia;
* **selectforeground** - kolor zaznaczonego tekstu;
* **show** - kontroluje sposób wyświetlania zawartości kontrolki. Gdy kontrolka zawiera jakiś tekst, jest on zastępowany podanym tutaj znakiem, typowym zastosowaniem tej opcji jest ukrywanie wpisywanego przez użytkownika hasła. W takim przypadku opcja ta powinna zostać ustawiona na **"\*"**;
* **state** - kontroluje stan kontrolki, który może być ustawiony na: **NORMAL**, **DISABLED** (to samo co "readonly"). Domyślna wartość to **NORMAL**;
* **takefocus** - określa, czy użytkownik może użyć tabulatora, by przekazać fokus tej kontrolce;
* **textvariable** - powiązane z obiektem klasy **StringVar** dostępnej w module **tkinter**;
* **validate** - określa kiedy zostanie wywołana funkcja z opcji **validatecommand** kontrolki. Możesz użyć jako ustawienia wartości **"focus"**, by wywołanie funkcji było wykonywane za każdym razem, gdy kontrolka otrzymuje lub traci fokus, **"focusin** by wywołanie funkcji było tylko, gdy kontrolka otrzyma fokus, **"focusout"** gdy kontrolka traci fokus, **"key"** gdy jakakolwiek modyfikacja nastąpi i **ALL** dla wszystkich tych sytuacji. Domyślnie ustawienie to **NONE** (brak wywołania funkcji);
* **validatecommand** - funkcja lub metoda, która zostanie wywołana by sprawdzić poprawność wpisywanego tekstu. Funkcja powinna zwracać **True**, gdy wpisany tekst spełnia warunki, **False** w przeciwnym razie;
* **vcmd** - to samo co **validatecommand**;
* **width** - szerokość kontrolki w jednostkach średniej szerokości liter tekstu;
* **xscrollcommand** - używane w celu połączenia kontrolki z kontrolką horyzontalnego paska przewijania. Ta opcja powinna być ustawiona by ustawić metodę w odpowiadającej kontrolce scrollbar.

**Button**

* **activebackground** - kolor tła przycisku, gdy ten jest wciśnięty;
* **activeforeground** - kolor tekstu, gdy kontrolka jest aktywna;
* **anchor** - określa, gdzie na przycisku powinien być umieszczony tekst (lub obraz). Możliwe ustawienia to **N** (north), **NE** (north-east), **E** (east), **SE** (south-east), **S** (south), **SW** (south-west), **W** (west), **NW** (north-west);
* **background** - kolor tła;
* **bg** - to samo co **background**;
* **bitmap** - bitmapa do wyświetlania na kontrolce. Gdy opcja **image** jest ustawiona to ta pozycje jest pomijana;
* **borderwidth** - szerokość obramowania przyciusku;
* **bd** - to samo co **borderwidth**;
* **command** - funkcja lub metoda, która zostanie wywołana, gdy przycisk zostanie wciśnięty;
* **compound** - kontroluje jak tekst i grafika są wyświetlane razem na przycisku. Domyślnie, jeżeli opcja **image** lub **bitmap** są ustawione, to tekst nie jest rysowany. Jeżeli opcja ta jest ustawiona na **CENTER** to tekst zostanie wyświetlony ponad grafiką, zaś ustawienie **BUTTON**, **LEFT**, **RIGHT** lub **TOP** sprawia, że tekst jest rysowany nad, po lewej, po prawej lub pod grafiką. Domyślne ustawienie to **NONE**;
* **cursor** - określa, jaki kursor ma być wyświetlany, gdy myszka znajdzie się nad kontrolką;
* **default** - gdy ustawione, to kontrolka jest ustawiona jako domyślny przycisk. Domyślnie ustawione na **DISABLED**;
* **disabledforeground** - kolor tła, gdy przycisk jest nieaktywny
* font - określa krój czcionki;
* **foreground** - kolor tła;
* **fg** - to samo co **foreground**;
* **height** - wysokość przycisku, gdy ten wyświetla tekst to wyrażona jest w jednostkach wysokości tekstu, w przeciwnym razie w pikselach;
* **highlightbackground** - kolor obramowania, gdy kontrolka nie ma fokusa;
* **highlightcolor** - kolor obramowania, gdy kontrolka ma fokus;
* **highlightthickness** - szerokość obramowania podświetlenia;
* **image** - obraz do wyświetlenia na kontrolce;
* **justify** - wyrównanie tekstu. Dostępne opcje: **LEFT**, **RIGHT** lub **CENTER**. Domyślna wartość to **CENTER**;
* **overrelief** - określa sposób zachowania przycisku, gdy kursor myszy znajdzie się nad nim. Dostępne opcje to **FLAT**, **GROOVE**, **RAISED**, **RIDGE**, **SOLID** lub **SUNKEN**;
* **padx** - dodatkowe poziome odsunięcie od tekstu lub obrazu wyświetlanego na przycisku;
* **pady** - dodatkowe pionowe odsunięcie od tekstu lub obrazu wyświetlanego na przycisku;
* **relief** - dekoracja obramowania, zazwyczaj ustawiona na **SUNKEN**, gdy przycisk jest wciśnięty i **RAISED** w przeciwnym przypadku. Inne możliwe ustawienia to **GROOVE** i **FLAT**. Domyślna wartość to **RAISED**;
* **state** - stan przycisku, możliwe ustawienia: **NORMAL**, **ACTIVE** lub **DISABLED**. Domyślna wartość to **NORMAL**;
* **takefocus** - określa, że przycisk otrzymuje fokus;
* **text** - tekst wyświetlany na przycisku;
* **textvariable** - zmienna powiązana z klasą **StringVar** z modułu tkinter;
* **underline** - ustawia podkreślenie tekstu. Domyślnie -1, co oznacza, że tekst nie jest podkreślany;
* **width** - długość przycisku wyrażona w szerokości średniej tekstu, gdy w kontrolce wyświetlany jest tekst, natomiast w pikselach, gdy wyświetlany jest obraz;
* **wraplength** - określa, czy tekst ma zostać literka po literce wyświetlany w poziomie. Domyślnie ustawione na 0, co oznacza wyświetlanie tekstu w jednym ciągu

**Kontrolka Text**

* **bg** - kolor tła;
* **bd** - szerokość obramowania kontrolki;
* **cursor** - kursor, który się pojawi gdy wskaźnik myszki znajdzie się nad kontrolką;
* **exportselection** - eksportuje zaznaczenie do schowka, aby tego uniknąć trzeba ustawić tą opcję na 0
* **font** - ustawia czcionkę (należy podać jako krotkę zawierającą nazwę kroju oraz rozmiar czcionki
* **fg** - domyślny kolor czcionki wyświetlanego tekstu;
* **height** - wysokość wyrażona w liniach tekstu. Ta wartość odnosi się do aktualnych ustawień czcionki;
* **highlightbackground** - kolor obramowania kontrolki, gdy ta nie ma fokusa;
* **highlightcolor** - kolor podświetlenie kontrolki, gdy ta ma fokus;
* **highlightthickness** - grubość obramowania kontrolki dla wyświetlania fokusa;
* **insertbackground** - kolor karetki kursora oznaczającego miejsce wpisywania tekstu;
* **insertborderwidth** - rozmiar obramowania 3W wokół karetki kursora wpisywania tekstu. Domyślna wartość to 0;
* **insertofftime** - liczba określająca w milisekundach czas znikania kursora karetki wpisywania tekstu. Domyślna wartość to 300 milisekund;
* **insertontime** - liczba określająca w milisekundach czas na jaki kursor karetki wpisywania tekstu się pojawia. Domyślna wartość to 600 milisekund;
* **insertwidth** - szerokość kursora karetki. Domyślna wartość to 2 piksele;
* **padx** - odsunięcie tekstu od prawej i lewej krawędzi kontrolki wyrażona w pikselach;
* **pady** - odsunięcie tekstu od górnej i dolnej krawędzi kontrolki wyrażona w pikselach;
* **relief** - - dekoracja obramowania, zazwyczaj ustawiona na **SUNKEN**, gdy przycisk jest wciśnięty i **RAISED** w przeciwnym przypadku. Inne możliwe ustawienia to **GROOVE** i **FLAT**. Domyślna wartość to **RAISED**;
* **selectbackground** - kolor tła zaznaczonego tekstu;
* **selectborderwidth** - szerokość obramowania rysowanego wokół zaznaczonego tekstu;
* **spacing1** - ta opcja określa ile dodatkowego miejsca jest dodawanego powyżej każdej linii tekstu. Jeżeli linia jest łamana to dodatkowa przestrzeń jest dodawana tylko ponad pierwszą linijką tekstu. Domyślnie ustawione na zero;
* **spacing2** - określa ile dodatkowej przestrzeni ma zostać dodanej pomiędzy łamanymi liniami tekstu. Domyślna wartość to zero;
* **spacing3** - określa ile dodatkowej przestrzeni ma zostać dodanej poniżej każdej linii tekstu. Jeżeli tekst jest łamany dodatkowa przestrzeń jest dodawana poniżej ostatniej linii. Domyślna wartość to zero;
* **state** - ustawienie stanu kontrolki. Możliwe opcje to: **NORMAL** (domyślne) lub **DISABLED**;
* **width** - szerokość kontrolki wyrażona w średniej szerokości znaków bieżącej czcionki;
* **wrap** - kontroluje, czy linie, które są zbyt długie aby zmieścić się w kontrolce mogą być wyświetlana. Domyślnie ustawione na **WORD** - co oznacza łamanie linii tekstu po ostatnim zdaniu, które się w danej linii mieści. Ustawienie na **CHAR** spowoduje łamanie tekstu na dowolnym pierwszym znaku, który się nie mieści w kontrolce;
* **xscrollcommand** - by tekst kontrolki był poziomo skrolowany, należy ustawić tą opcję na metodę **set()** kontrolki typu **Scrollbar**;
* **yscrollcommand** - by tekst kontrolki był pionowo skrolowany, należy ustawić tą opcję na metodę **set()** kontrolki typu **Scrollbar**;

**Kontrolka Checkbox**

Opcje, które już wcześniej nie zostały opisane:

* **command** - funkcja, która zostanie wywołana przy wciśnięciu przycisku;
* **offvalue** - ustawienie wartości, gdy przycisk jest odznaczony. Domyślna wartość do zero;
* **onvalue** - ustawienie wartości, gdy przycisk jest zaznaczony. Domyślna wartość to 1;
* **value** - przypisanie zmiennej, która będzie ustawiana za każdym razem, gdy stan przycisku się zmieni. Ta zmienna może być obektem klasy **IntVar** lub **StringVar**.

**Kontrolka Radiobutton**

Lista ważniejszych dla tej kontrolki opcji:

* **command** - podpinanie funkcji w odpowiedzi na kliknięcie przycisku;
* **value** - wartość, dla której kontrolka jest ustawiona w stan zaznaczenia;
* **variable** - zmienna sterująca zaznaczeniem, będąca obiektem klasy **StringVar** z modułu **tkinter**. Gdy zawartość zmiennej jest równa wartości zapisanej w ustawieniu **value**, to kontrolka jest zaznaczona, w przeciwnym przypadku odznaczona;
* **text** - tekst wyświetlany w kontrolce

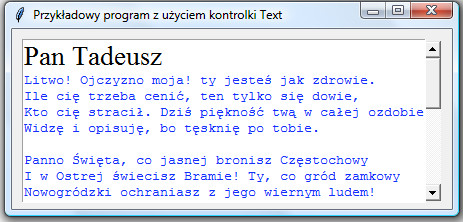
**Kontrolka Listbox**

* **background**
* **bd**;
* **bg**
* **borderwidth**
* **cursor**
* **exportselection**
* **fg**
* **font**
* **foreground**
* **height**
* **highlightbackground**
* **highlightcolor**
* **highlightthickness**
* **relief**
* **selectbackground**
* **selectborderwidth**
* **selectforeground**
* **selectmode**
* **setgrid**
* **takefocus**
* **width**
* **xscrollcommand**
* **yscrollcommand**
* **listvariable**

**Kontrolka Scrollbar**

Kontrolki tego typu to paski przewijania, które jako takie mogą być podpinane pod kontrolki takie jak **Text** czy **Entry**. Przykład zastosowania kontrolki **Scrollbar** jako paska przewijania poziomego dla kontrolki typu Scrollbar.

1. import tkinter as tk
2. window = tk.Tk() # tworzenie okna głównego programu
3. window.geometry(***"440x180"***) # ustawienie wymiarów okna
4. window.title(***"Przykładowy program z użyciem kontrolki Text"***) # ustawienie tytułu okna
5. sb\_textbox = tk.Scrollbar(window) # tworzenie kontrolki paska przewijania
6. textbox = tk.Text(window, width = 50, height = 10, yscrollcommand = sb\_textbox.set) # tworzenie textbox-a i podpinanie pod yscrollcommand metody set z paska przewijania
7. textbox.pack()
8. sb\_textbox.place(in\_ = textbox, relx = 1., rely = 0, relheight = 1.) # ustawienie miejsca oraz wymiarów paska przewijania (relatywnie w odniesieniu do wymiarów kontrolki textbox-a
9. textbox.place(x=10, y=10)
10. textbox.insert(tk.END, ***"Pan Tadeusz"***, (***"h1"***)) # tekst z formatowaniem zapisanym w znaczniku h1, który nieco później zostanie utworzony
11. textbox.insert(tk.END, ***"n"***)
12. textbox.insert(tk.END, ***""***"Litwo! Ojczyzno moja! ty jesteś jak zdrowie.
13. Ile cię trzeba cenić, ten tylko się dowie,
14. Kto cię stracił. Dziś piękność twą w całej ozdobie
15. Widzę i opisuję, bo tęsknię po tobie.
16. Panno Święta, co jasnej bronisz Częstochowy
17. I w Ostrej świecisz Bramie! Ty, co gród zamkowy
18. Nowogródzki ochraniasz z jego wiernym ludem!
19. Jak mnie dziecko do zdrowia powróciłaś cudem
20. (Gdy od płaczącej matki pod Twoją opiekę
21. Ofiarowany, martwą podniosłem powiekę
22. I zaraz mogłem pieszo do Twych świątyń progu
23. Iść za wrócone życie podziękować Bogu),
24. Tak nas powrócisz cudem na Ojczyzny łono.
25. Tymczasem przenoś moję duszę utęsknioną
26. Do tych pagórków leśnych, do tych łąk zielonych,
27. Szeroko nad błękitnym Niemnem rozciągnionych;
28. Do tych pól malowanych zbożem rozmaitem,
29. Wyzłacanych pszenicą, posrebrzanych żytem;
30. Gdzie bursztynowy świerzop, gryka jak śnieg biała,
31. Gdzie panieńskim rumieńcem dzięcielina pała,
32. A wszystko przepasane, jakby wstęgą, miedzą
33. Zieloną, na niej z rzadka ciche grusze siedzą.""", ("p")) # dodawanie tekstu bez dodatkowego formatowania
34. textbox.tag\_add("h1", "1.0", "1.0") # dodanie nowego tagu o nazwie h1 (jak nagłówek główny w HTML-u), drugi i trzeci argument to początek zaznaczenia i koniec zaznaczenia (w tym przypadku wybrałem, że zaznaczenia nie będzie 1 - oznacza pierwszy wiersz a liczba po kropce określa numer znaku w danym wierszu
35. textbox.tag\_config("h1", font=("Times New Roman", 20)) # formatowanie tekstu: kolor tła; kolor tekstu; czcionka podana jako krotka z: nazwą kroju, rozmiarem
36. textbox.tag\_add("p", "1.0", "1.0")
37. textbox.tag\_config("p", foreground="#1130ff")
38. sb\_textbox.config(command = textbox.yview) # podpinanie pod kontrolkę paska przewijania metody yview textbox-a
39. window.mainloop() # wywołanie głównej pętli programu



**Rys. 2**

**Kontrolka Scale**

Ta z kolei kontrolka przypomina nieco kontrolkę **Scrollbar**, jednakże ta kontrolka służy do ustawiania i wyświetlania wartości liczbowych. Oto przykład kodu tworzącego taką kontrolkę:

1. scale = tk.Scale(window, from\_ = 0, to = 10, orient=tk.HORIZONTAL) # tworzę kontrolkę typu Scale przyjmującą wartości całkowite od 0 do 10 i ustawioną na ułożenie poziome
2. scale.pack()
3. variable = tk.StringVar()
4. label = tk.Label(window, textvariable = variable) # to będzie kontrolka pomocnicza
5. label.pack()
6. def scale\_changed(ch):
7. variable.set(ch) # tu będę wyświetlał dane, mógłbym je wyciągnąć również korzystając z metody get: scale.get()
8. scale.config(command = scale\_changed) # podpinam funkcję scale\_changed pod zdarzenie wywoływane, gdy wartość ustawienia kontrolki została zmieniona

Widok kontrolki Scale z modułu tkinter języka programowania Python

**Matplotlib**

import numpy as np

import matplotlib.pyplot as plt

from pylab import show, arange, sin, plot, pi

x = np.arange(0, 5, 0.1)

y = np.sin(x) #(numpy - funkcja sin)

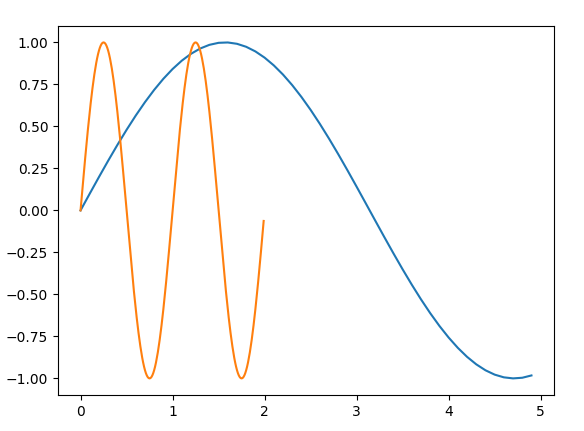
plt.plot(x, y) # rysowanie (biblioteka matplotlib)(niebieski)

t = np.arange(0.0, 2.0, 0.01)

s = sin(2 \* pi \* t) #(numpy - funkcja sin)

plot(t, s) # pomaranczowy

show() # pokazuje nasze okno z wykresem



from matplotlib.figure import Figure

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import tkinter as tk

fig = Figure(figsize=(12,8), facecolor = "grey") # obiekt (prostokąt 12 x 8 pik) - prostokąt na którym umieścimy wykres

axis = fig.add\_subplot(211) # rozmieszczenie naszego wykresu w oknie fig (211- 2 to dwa rzędy w oknie, 1 to jedna kolumna, 1 to umieszczenie wykresu w 1 rzędzie )

xValues = [1,2,3,4] # wartości na naszym wykresie

yValues = [5,7,6,8]

axis.plot(xValues, yValues) # drukowanie wartości na wykresie

axis.set\_xlabel("Horizontal Label") # etykiety naszych osi

axis.set\_ylabel("Vertical Label")

axis.grid(linestyle=":") # typ linii ('-', '--', '-.', ':', 'None', ' ', '', 'solid', 'dashed', 'dashdot', 'dotted')

def \_destroyWindow(): # funkcja zamknięcia naszego wykresu/okna

    root.quit()

    root.destroy()

root = tk.Tk() # obiekt głównego okna bibliteki tkinter

root.configure(background='black')

root.withdraw() #

root.protocol("WM\_DELETE\_WINDOW", \_destroyWindow) # nasza akcja jęsli będziemy chcieli zamknąc wykres/okno

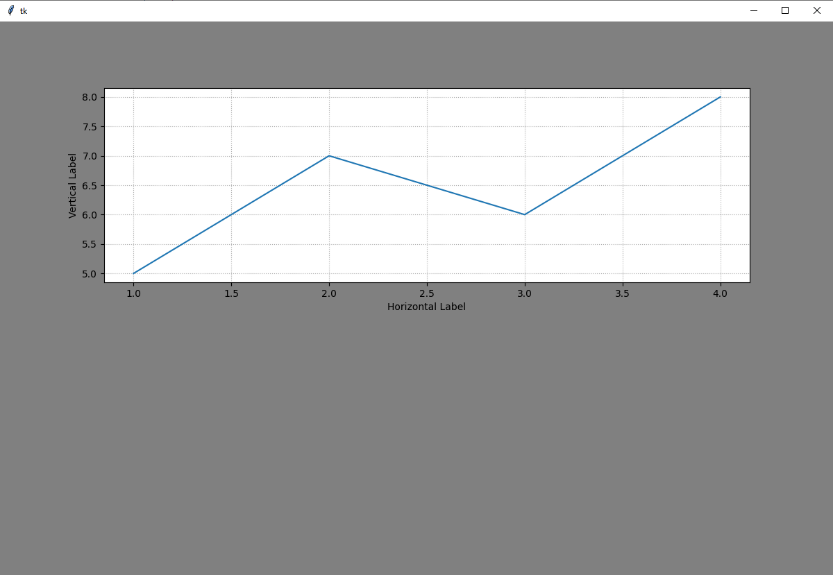
canvas = FigureCanvasTkAgg(fig, master=root) # umieszczamy

nasze okno wykresu fig na naszym głównym oknie root

canvas.\_tkcanvas.pack(side=tk.TOP, fill=tk.BOTH, expand=1) # ułożenie naszego okna w głównym oknie

root.update()

root.deiconify()

root.mainloop() ****

from matplotlib.figure import Figure

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import tkinter as tk

fig = Figure(figsize=(12,5), facecolor = "grey") # obiekt

axis = fig.add\_subplot(111)

axis.set\_facecolor("black") # kolor tła naszego pola wykresu

axis.set\_title("Wykres 1",fontsize=20,color="white")#tytuł wykr

axis.grid(linestyle=":", color="white) # typ, kolor linii wykr.

# axis.set\_axis\_off() # brak osi

# axis.set\_ylim(5,10) # skalowanie osi y(oś przyjmie wartości między 5 a 10)(axis.set\_xlim())

xValues = [1, 2, 3, 4] # wartości na naszym wykresie

yValues0 = [5.7, 7.5, 6.6, 9]

yValues1 = [5.5, 100, 8.5, 9]

yValues2 = [6.5, 7.1, 8.7, 9]

# dynamiczne skalowanie wykresu -------------------------------

yAll = [yValues0, yValues1, yValues2] # lista list wartości

minY = min([y for yValues in yAll for y in yValues ]) # szukamy minimalnej  wartości y na naszym wykresie

yUpperLimit = 20 # nasz limit dla mksymalnej wartości y

maxY = max([y for yValues in yAll for y in yValues if y < yUpperLimit]) # szukamy maksymalnej wartości y na naszym wykresie , ale nie większejniż  limitem yUpperLimit

# maxY = (max([y for yValues in yAll for y in yValues if y < yUpperLimit])) + (max([y for yValues in yAll for y in yValues if y < yUpperLimit])) \*0.05 # to samo co wyżej tylko 5 % więcej

axis.set\_ylim(minY,maxY)

axis.set\_xlim(min(xValues),max(xValues))

# koniec dynamicznego skalowania ------------------------------

# axis.bar(xValues,yValues0) # wykres słupkowy

axis.scatter(xValues,yValues0, marker='o', alpha=1, color="yellow",edgecolors="orange",) # wykres scatter (same punkty)(marker="x")(umieszczony na wykresie pierwszym od góry)

# axis.pie([5,10,22,34], labels=['10', '5', 'c', "asdfg"]) # wykres tortowy(kołowy)

t0, = axis.plot(xValues, yValues0, color="green") # drukowanie wartości na naszym wykresie. Color=green – kolor lini wykresu

t1, = axis.plot(xValues, yValues1, color="red") # t0, - przecinek jest konieczny, jeżeli chcemy dodać legendę

# axis.legend()

t2, = axis.plot(xValues, yValues2, color="blue")

axis.set\_xlabel("Horizontal Label", color="white", fontsize=20) # etykiety naszych osi

axis.set\_ylabel("Vertical Label", fontsize=16, fontstyle="italic")

fig.legend((t0, t1, t2), ("First line", "Second line", "third line"),"upper right") # dodanie legendy w obiekcie fig # t0 ,t1 ,t2- nasze krzywe do opisania,upper right –rozmieszczeń Legendy

# 'best', 'upper right', 'upper left', 'lower left', 'lower right', 'right', 'center left', 'center right', 'lower center', 'upper center', 'center'

**Końcówka skryptu jak wyżej**

