**Ustawienia pliku, start**

|  |  |
| --- | --- |
| #-\*-coding: utf-8-\*- | polskie znaki, kodowanie Unikod UTF-8 przy zapisywaniu, dodać komentarz jaki przyjęto tryb kodowania w pliku |

**Listy, krótkie, słowniki, tablice z haszowaniem**

|  |  |
| --- | --- |
| zmienna  STAŁA  dir(polecenie)  #komentarz  ‘’’ ‘’’ lub ””” ”””  \n  \"  Lista = [a,b,c]  Krotka = (a,b,c,d)  slownik = {'A1' : 'AA1','B2' : 'BB2'}  slownik = {'Imie':['Ania','Michał','Przemek'], 'Wiek':[18,25,40]}  slownik['B2']  slownik.add(‘aaa’)  slownik.update(dodawane\_elementy)  słownik.remove(usuwane\_elementy)  slownik["klucz"]  slownik.get("klucz", "tekst jak nie znajdzie klucza")  słownik.values()  słownik.items()  zmienna.upper()  zmienna.lower()  if zmienna isupper():  zmienna.index("a")  lista = lista + [6]  lista.append(6)  lista.append(“nowy element listy”)  lista.extend(druga\_lista)  nowa\_lista = stara\_lista.copy()  lista.insert(nr\_indeksu\_gdzie, "nowy\_element")    lista.clear()  lista.count("element")  lista.sort()  lista.reverse()  lista.insert(miejsce,"element")  lista[miejsce] = element  tekst.remove("co")  lista.pop()  del lista[które\_elementy]  tekst.replace("co", "naco")  tekst =’ciag znakow {} oraz {}  tekst.format  print(tekst.format(zm1, zm2))  definiujemy hash\_table = {}  if a in hash\_table:  hash\_table [znaczek] = 1  hash\_table [znaczek] += 1 | Piszemy małymi literami  Piszemy DUZYMY LITERAMI  wypisuje co można z danym poleceniem można zrobić i jaka jest składnia  komentarz wiele linii, trzy średniki  przejście do nowej linii w tekście  po ukośniku każdy znak jest traktowany jako tekst  lista – definicja w nawiasach  krótka to lista, której nie można zmieniać  słownik, nawiasy, elementy rozdzielane dwukropkiem  zapytanie o element słownika  dodanie elementu do słownika  dodanie listy elementów  usunięcie elementów ze słownika  zwraca wartość klucza ze słownika  zwraca wartość klucza ze słownika lub komunikat  same wartości słownika pozycje słownika w postaci kluczy i wartości  zmiana na DUŻE LITERY  zmiana na małe litery  sprawdza czy duża LITERA  zwraca indeks/numer litery w tekście  dodanie elementu na koniec listy  dodaje do listy na końcu drugą listę  kopiuje wartości do nowej listy  wstawia w miejscu nowy element li.insert(2,’aa’)  usuwa wszystkie elementy z listy  liczy ile jest podanych elementów w liście  sortuje rosnąco listę  podaje listę od końca  dodanie elementu w miejscu lista.insert(2,’aa’)  zamiana elementu na danym miejscu  usunięcie wskazanego tekstu  usuwa ostatni element listy  usuwa elementy z listy, np. del.lista[2]  zmiana elementów co w naco  wpisanie w tekst wartości zmiennych  dodawanie elmentu do tablicy/słownika, w nawiasie 1 kolumna po równa się wartość  zwiększanie elementu słownika |

**Wyświetlanie na ekran, zczytywanie od użytkownika**

|  |  |
| --- | --- |
| print (zawartosc)  print('\n')  print(zawartosc, end=' ')  print('a','b','c',sep='……')  input() input(“Podaj wartość”) | wyświetlenie na ekran  pusta linia, pierwsze użycie to przejście do kolejnej linii  wartości rozdzielane znakami między apostrofami ‘znak’  separator między elementami  wprowadzenie wartości przez użytkownika |

**Zakresy zmiennej**

|  |  |
| --- | --- |
| "tekst" lub 'tekst'  tekst[indeks element lub zakres]  [0]  [-1]  [-3]  [:]  [:N]  [M:]  [N:M]  [::2]  [::-1]  str (zmienna\_numeryczna)  int (zmienna\_tekstowa)  float (zmienna\_tekstowa)  from math import \*  floor (liczba)  ceil (liczba)  sqrt (liczba) | ciąg tekstowy  określenie pozycji w liście  pierwszy element  ostatni element  trzeci od końca  wszystkie elementy  wszystkie elementy od 0 do N-1  wszystkie elementy od M (uwgz. Indeks 0)  elementy z zakresu N do M (start 0, wyś. M-1)  wszystkie co 2  wszystkie od końca do początku  zamienia na string  zamienia na liczbę całkowitą  zamienia na liczbę rzeczywistą z przecinkiem  zaokrągla do całkowitej w dół  zaokrągla do całkowitej w górę  pierwiastek kwadratowy |

**Operacje na tekstach, zmiennych i listach**

|  |  |
| --- | --- |
| + - \* /  a+=1 -= \*= /=  //  % np. i%2  2\*\*3 lub pow(2,3)  min(1,2,3,4,-2)  max(1,8,-10,5)  abs(-2)  round(liczba,dokładność)  len(tekst)  ‘ciag’ in tekst  ‘ciag’ not in tekst | operacje na liczbach  zmiana wartości zmiennej  dzielenie do wartości całkowitych  dzielenie, podanie reszty z dzielenia  potęgowanie 2^n  zwraca wartość minimalną  zwraca wartość maksymalną ze zbioru  zwraca wartość bezwzględną |-2|  zaokrąglenie z zadaną dokładnością, 2-miejsc po przecinku, -1 do pełnych 10, -2 do pełnych 100 itd., bez dokładności - wówczas z domyślną  długość zmiennej, np. ilość znaków  sprawdza zawartość, wyrzuca True/False  sprawdza brak zawartości, wyrzuca True/False |

**Warunki, pętle**

|  |  |
| --- | --- |
| if warunek: zdarzenie  elif warunek:zdarzenie  else:  == !=  &  |  and or and not not domyślnie True  for liczba in range(1,11):  for liczba in range(1,11,3):  for element in lista:  while warunek:  while True:  break  najmniejsza = None | prosty warunek  operatory równy, różny  zamiast and  zamiast or  łączenie warunków, lub  pętla, wartości od 1 do 10 bez ostatniej  pętla, krok co 3  przechodzi przez całą listę  pętla while  wyjście z pętli  zmianna pusta None |

**Funkcje**

|  |  |
| --- | --- |
| def NazwaFunkcji (parametr1, parametr2):  return  return zmienna  nowaZmienna = wynikFunkcji (a,b)  lambda <parametry> : <wyrażenie> | definiowanie funkcji  zakończenie funkcji  wynik funkcji jako zmienna  funkcja lambda |
| sorted()  parametry: key i reverse  reverse=True (odmyślne False)  Key , np. key=str.lower | Funkcja do sortowania, zwraca listę, jako argument przyjmuje obiekt iterowany  określa kolejność  przyjmuje funkcję, według której ma posortować sekwencję. |
| split() | Dzieli tekst na wyrazy, np. „a b c”.split() |
| lambda <parametry> : <wyrażenie>  lambda a,b: a+b | funkcja lambda  Funkcja nienazwana, stosowana tam, gdzie jako argumentu, czy parametru potrzebujemy funkcji  po lambda przypisujemy zmienne dla argumentów, a po dwukropku znajduje się ciało funkcji |

**Różne biblioteki, wykorzystanie modułów**

|  |  |
| --- | --- |
| import time  time.sleep(1)  import random  random.choice(["orzeł", "reszka"])  import nazwa\_pliku  nazwa\_pliku.nazwa\_funkcji  from nazwa\_pliku import nazwa\_funkcji  nazwa\_funkcji | import biblioteki time  oczekiwanie 1 sek  losowanie z listy  wykorzystanie z modułów z innych plików  import mod1  print (mod1.zlicz("Ala ma kota"))  można zaimportować i używać nazw bez nazw plików  From mod1 import zlicz  x = zlicz ("Nowy sposób importu modułu") |
| $ pip install <nazwa pakietu>  pip install xlwt openpyxl xlrd  pd.read\_excel('c:/users/mo1de/Documents /GitHub/mo1der/imiona.xlsx')  pd.read\_excel('ścieżka', sheet\_name= 'nazwa arkusza', header=None, names=['Nazwy', 'Poszczególnych', 'Kolumn'])  read\_csv()  read\_excel()  read\_json()  read\_pickle()  read\_sql()  to\_csv()  to\_excel()  to\_json()  to\_pickle()  to\_sql()  to\_html() | pip jest programem, który może zainstalować wskazane przez nas pakietów, zawierających moduły, które następnie możemy zaimportować poleceniem import  xlwt zapis xls  xlrd odrzyt xls i xlsx  openpyxl zapis do plików xlsx  Listę wszystkich modułów, które możemy zainstalować znajdziemy na https://pypi.org |

**Programowanie obiektowe**

|  |  |
| --- | --- |
| Definiowanie klasy  class nazwa\_klasy: np. człowiek  imie = ""  wiek = None  def przywitaj(self):  print ("Cześć. Mam na imię", self.imie)  piotr = człowiek()  piotr.wiek = 12  konstruktor  mike = człowiek(„Mike”,45,”mężczyzna”)  def \_\_init\_\_(self, p1,p2,p3):  self.imie = p1  self.wiek = p2  self.plec = p3  def \_\_init\_\_(self, imie,wiek,plec):  self.imie = imie  self.wiek = wiek  self.plec = plec | Wewnątrz naszej klasy, znajduje się  definicja trzech zmiennych, imię, wiek oraz płec,  funkcje, które obiekt takiej klasy, powinien mieć.  ’self’. Jest on przekazywane, jako parametr, do każdej z naszych funkcji, a następnie używane, aby odwołać się do zmiennych, zdefiniowanych w klasie.  funkcja / metoda, klasy, która jest wywoływana automatycznie w momencie jej utworzenia. Co więcej, konstruktor, również może przyjmować parametry, i te parametry, to jest to, co znajduje się w momencie utworzenia obiektu.  Konstruktor zawsze nazywa się '\_\_init\_\_’. Jako parametry, przyjmuje 'self’, a następnie parametry przekazywanie w momencie tworzenia obiektu.  Dwie spacje przy init  Metody klasy to zdefiniowane funkcje def |
| Dziedziczenie  klasa ptak, ma kolor, szybkość, wielkość, lata, wydaje odgłosy  klasa sowa, potomek klasy ptak, dodatkowo czuwa w nocy  klasa orzeł, potomek klasy ptak, dodatkowo poluje  class orzeł (ptak):  def wydajOdgłos(self):  pass  class ptak:  def \_\_init\_\_ (self, gatunek, szybkość):  self.gatunek = gatunek  self.szybkość = szybkość  class orzeł (ptak):  def \_\_init\_\_ (self, szybkość):  super().\_\_init\_\_("orzeł", szybkość)  Klasy i metody abstrakcyjne  from abc import ABC, abstractmethod  class ptak(ABC):  @abstractmethod  def wydajOdgłos(self):  pass | znacząco może nam uprościć definicje klas i pozwolić na skupieniu się, tylko na tych aspektach, które faktycznie te klasy charakteryzują.  Pusta metoda/funkcja  Konstruktor w potomku  utworzenie dodatkowego konstruktora, już w samym potomku, który dla orła będzie przypisywać 'gatunek = orzeł’, tak aby nie trzeba było tego robić przy tworzeniu obiektu  Wywołane konstruktory klas potomnych, inicjują konstruktor klasy od której dziedziczą, za pomocą funkcji super ()  Jeżeli klasa, posiada metody abstrakcyjne, oznacza to tyle że, potomstwo jest zmuszone ją nadpisać. Równocześnie klasa taka, nigdy nie powinna być sama w sobie wykorzystana do tworzenia obiektów.  importujemy 'ABC’ oraz 'abstractmethod’, z pakietu abc. Jest to pakiet, który umożliwia nam proces tworzenia abstrakcyjnych klas i metod  Klasa ptak, dziedziczy ABC  Metody, które są poprzedzone (są za) '@abstractmethod’, stają się metodami abstrakcyjnymi, które muszą zostać nadpisane przez potomstwo. Samo wyrażenie '@abstractmethod’, jest czymś co nazywamy dekoratorem.  Klasy 'orzeł’ oraz 'pingwin’, są zmuszone nadpisać metody klasy 'ptak’, oznaczone jako '@abstractmetod’.  W praktyce, wszystkie metody, klasy która jest dziedziczona mogą być oznaczone dekoratorem '@abstractmethod’. W ten sposób możemy wymusić, aby gatunki ptaków miały te same metody, ale ich implementację zostawić już dla nich samych. Co w praktyce, okazuje się być bardzo praktyczne. |

**Pandas**

|  |  |
| --- | --- |
| pip install pandas  import pandas as pd | istalacja bibioteki pandas  importujemy do wykorzystania bibliotekę pod standardową nazwą jaką nazywa się pandas - pd |
| NaN | Oznaczenie pustej komórki |
| read\_csv() jest to możliwe korzystając z dodatkowych parametrów takich jak:  dtype, który zdefiniuje nam od razu typy kolumn,  skiprows, który umożliwi nam ominięcie pierwszego, problematycznego wiersza.  usecols, gdzie możemy podać interesujące nas kolumny  czy też converters, który umożliwia nam zastosowanie apply  def zamien(wartosc):  if wartosc == 'No': return False  if wartosc == 'Yes': return True  films = pd.read\_csv('https://analityk.edu.pl/wp-content/uploads/2020/12/film.csv',  sep=';',  encoding = "ISO-8859-1",  skiprows=[1],  dtype={'Length':'float64', 'Popularity':'float64'},  usecols=['Year','Length','Title','Subject','Popularity','Awards'],  converters={'Awards':zamien})  converters={'Awards':lambda x: True if x == 'Yes' else False}) | **Zaczytanie danych z pliku** |
| tabela.columns=["styczeń","luty","marzec", "kwiecień"] | wpisanie nazw kolumn |
| s = pd.Series([11,33,55,99]) | typ series, zawiera kolumnę z indeksem jak ją tworzymy |
| tabela = pd.DataFrame()  arkusz=pd.DataFrame(lista)  age = pd.DataFrame({ 'Name':['Michał', 'Adam', 'Ewa', 'Jakub'],  'Age':[39,28,19,32]})  nazwa\_arkusza.shape  nazwa\_arkusza.info()  nazwa\_arkusza.describe()  nazwa\_arkusza.nazwa\_kolumny.describe()  df.dtypes  nazwa\_tabeli.count()  tabela.head()  tabela.tail()  tabela[2:10]  tabela['nazwa\_kolumny\_do\_wyświetlenia']  nazwa\_arkusza[0:8][['nazwy\_kolumn\_rozdzielane przecinkami']]  nazwa\_arkusz.nazwa\_kolumny  nazwa\_arkusza.nazwa\_kolumny.unique()  pd.unique(animals)  nazwa\_arkusza['nazwa\_kolumny'].unique()  nazwa\_arkusza.nazwa\_kolumny.is\_unique  nazwa\_arkusza.isnull()  nazwa\_arkusza.isnull().sum()  notnull()  duplicated()  drop\_duplicates()  nazwa arkusza.index  nazwa\_arkusza.set\_index('nazwa kolumny', inplace=True)  kostium = kostium.reset\_index()  kostium.reset\_index(inplace=True)  kostium.loc['Arizona']  kostium.loc['Arizona','1']  kostium.loc[3,’1′]  kostium.iloc[3,2]  kostium.iterrows()  for index, zawartosc in kostium.iterrows():  if zawartosc['1'] == 'Rabbit': | utworzenie pustego DataFrame (arkusza)  utworzenie DataFrame (arkusza) z listy  tworzenie tabel z palca  info ilość wierszy i kolumn  informacja na temat kolumn  statystyki dla kolumn z danymi  statystyki dla kolumny  typy danych w kolumnach  zliczenie wierszy  wyświetla pierwsze 5 rekordów  5 ostatnich rekrodów  wyświetla rekrody w zakresie, od 3 do 9 włącznie  nazwa kolumny mozę być w nawiazie i cudzysłowie lub po kropce  wartości unikalne  sprawdza czy kolumna zawiera wartości unikalne  sprawdza czy komórki zawierają wartości wynik True/False  suma komórek  nie puste  sprawdza czy są zduplikowane rekordy  usuwa zduplikowane rekordy  sprawdzenie indeksów  zmiana kolumny z indeksem liczbowym na inna kolumnę zawierającą unikalne wartości  resetowanie indeksu  wywołanie wiersza ze wskazanym indeksem  wywołanie dokładnie komórki  wywołanie komórki z indeksem numerycznym  wywołanie wyłącznie na podstawie indeksów numerycznych  zwraca kolejne wiersze  przechodzi przez kolejne pozycje  sprawdzenie warunku |
| **Maski, grupowanie**  maska na postawie True/False  kostium[ kostium['1'] == 'Rabbit' ]  &  |  nazwa\_arkusza[(filtr1) | (filtr2)]  nazwa tabeli[warunki].groupby(zasady wg czego grupowac).kolumna.funkcja | kostium['1'] == 'Rabbit'  zamiast and  zamiast or  łączenie masek  grupowanie |
| nazwa\_arkusza['nazwa\_nowej\_kolumny'] = 'wartosci\_nowej\_kolumny'  kostium.rename(columns={'1':'Pierwszy', '2':'Drugi'})  kostium.drop('Nowa', axis=1)  films = films.drop(0)  films = films.drop(columns=['\*Image'])  kostium['Połączone'] = kostium['3'] + ' | ' + kostium['4']  kostium[['Trzeci','Czwarty']] = kostium.Połączone.str.split('|',expand=True)  wiersz['Osoba'].split(' ')  def rozdziel(wiersz):  wiersz['Imie'], wiersz['Nazwisko'] = wiersz['Osoba'].split(' ')  return wiersz  df = df.apply(rozdziel, axis=1)  apply(nazwa funkcji)  dfr.Wyniktxt = dfr.Wynik.apply(abc)  df = df[['B', 'C', 'A']]  order = [1,2,3,0]  df = df[[df.columns[i] for i in order]]  cols = list(df.columns.values)  nazwa\_tabeli.count()  nazwa\_tabeli.groupby('nazwa kolumny').count()  nazwa\_tabeli.groupby(['nazwa kolumny','nazwa kolumny2']).count()  films.Length.mean()  film.groupby('Year').Popularity.mean()  films.Length = pd.to\_numeric(films.Length)  films.dtypes  films.Popularity = films.Popularity.astype('float64')  def zamien(wartosc):  if wartosc == 'No': return False  if wartosc == 'Yes': return True  films.Awards = films.Awards.apply(zamien)  films.dtypes  object  int64  float64  bool  Category  datetime  agg()  films.groupby('Year').agg({'Popularity':['min','max'], 'Length':['min','max']})  .agg(mininalna\_popularnosc=('Popularity','min'),  maksymalna\_popularnosc=('Popularity','max'))"  x=x.zmienna.apply(lambda x: 'Bardzo popularny' if x > 60 else 'Niszowy') | Dodanie nowej kolumny  Zmiana nazwy kolumn  Usunięcie kolumny  Usunięcie wiersza  Połączenie kolumn  Rozdzielenie kolumn  na podstawie znaku  Funkcja do rozdzielania  pozwala uruchomić funkcję dla każdej komórki  wpisanie w wartości kolumny, funkcję wykorzystującą dane z innej kolumny  Zmiana kolejności kolumn  Sprawdzenie kolejności kolumn  zliczenie wierszy  zliczanie wg klucza  średnia z danej kolumny  zmiana typu danych na inny  zmiana typu danych na inny apply  String (łańcuch znaków)  Integer (liczba całkowita)  Float (liczba rzeczywiste)  bool (True lub False)  NA  Datetime  Dodanie dodatkowych kolumn ze statystyką  zamiana wartości |
| Tabele przestawne  x = films.groupby(['Year','Subject']).agg(avg\_popularity = ('Popularity','mean'))  x.unstack()  x.unstack(fill\_value='')  pd.pivot\_table(films,  index='Year',  columns='Subject',  values='Popularity',  aggfunc='mean')  margins=True | wypełnia wartości puste znakami w nawiasie  Tabele przestawne  index – czyli która kolumna ma być po lewej stronie w wierszach, jako nasz index  columns – która kolumna ma utworzyć szereg nowych kolumn, na górze  values – wynik operacji na której kolumnie ma byś widoczny 'wewnątrz’ naszej tabeli przestawnej  aggfunc – jaką operację chcemy przeprowadzić – mean, sum, count, max, min itp  dodaje sumy |
| łączenie zbiorów  Merge, inner, czyli tylko część wspólna  pd.merge(age\_new, city, on='Kolumna\_wspolna', how='inner')  Merge, outer, czyli wszystkie dane, puste pola wypełniane NaN  pd.merge(age\_new, city, on='Name', how='outer')  Merge, left, czyli wszystkie dane z lewego zbiory danych  pd.merge(age\_new, city, on='Name', how='left')  Merge, right, czyli wszystkie dane z prawego zbiory danych  pd.merge(age\_new, city, on='Name', how='right') | dodanie nowych kolumn pasujących do klucza |

filter(), map(), reduce()

list

dlaczego przypisanie listy do innej listy utrzymuje obie wartości takie same, pomimo, że jedna się zmienia to dlaczego druga również

dlaczego nie wychodzi z pętli for komunikatem break, dla zagnieżdżonej pętli for