1

Saturday, March 4, 2023 7:38 PM

Riječnik , mapa, asocijativno polje

ime_mape = { ključ1 : vrijednost1, ključ2 : vrijednost2
vrijednost = ime_mape [ključ]

Liste
ime_liste = [vrijednost1, vrijednost2]
vrijednost = ime_liste [cijeli_broj]

'''
komentar u vise redova
'''
komentar u jednom redu

uvjetno grananje
```
if size < 0:
    raise Exception()
```

uvjetni operator
multiple = vrijednost_ako_je_true **if** usporedba **else** druga_vrijednost

raise Objekt_Tipa (raise ValueError('number must be non negative')

petlje

```
for x in iterator:
    naredba
    naredba
```

sintaksa
- potenciranje --> **, rad
- rad nad bitovima --> &, | , ^
- logicke --> and, or, not

a=a/5 <--> a/=5

formatiranje

'{0:.1f} {1}'.format(prva_realna_vrijednost, druga_vrijednost)

glavni_program

ime programskog modula

if __name__ == '__main__':
    naredba

tipovi podataka

type(123) --> vraca tip objekta
isinstance(123, int) --> ocigledno
False --> 0, None, [], {}
bool(0)

py3 --> 1/3 = 0.3333 , py2 = 1/3 = 0
cjelobrojno **dijeljenje** 3 // 2 = 1 , cjelobrojni **ostatak** 2 % 3 = 2
potenciranje ** = 2 ** 12 , 2 na 12

import fractions
a = **fractions.Fractions**(2,4)
b = fractions.Fractions(8,12)
a + b

c = **complex**(1, 3) // realni, imaginarni dio
d = complex(5, 8)

lista = [0, 1, 2]
lista = list()

lista[indeks]

l = ['a','b','c','d','e']
l[-1] --> 'e'
l[-2] --> 'd'
l[-3] --> 'c'
l[0] --> 'a'
l[1] -->'b'

raspon = lista indeks_pocetka : indeks_kraja

Vraća elemente od indeks_pocetka uključujući) do indeks_kraja isključujući
Ako se indeks_početka izostavi indeks_početka je 0
Ako se indeks_kraja izostavi indeks_kraja je duljina
Oba indeksa se mogu izostaviti vraća kopiju cijelog polja

spajanje lista
nova = lista1 + lista2

ne stvara se nova lista

dodavanje elementa na kraj
nova.append(100)

dodavanje elemenata liste na kraj druge
nova.extend([200,300])

umetanje elementa
nova.insert(1, 543)

broj elemenata
len([1,2,3])

broj pojavljivanja
[1,2,3].count(2)

ispitivanje prisutnosti elemenata
2 in [1, 2, 3]

index prvog pojavljivanja
[1,2,3].index(2)

uklanjanje n-tog elementa
del lista[3]

uklanja prvo pojavljivanje vrijednosti
[1,2,3,3,4].remove(3) = [1,2,3,4]

[1,2,3,4].pop() - zadnji ili n-ti element
[1,2,3,4].pop(2) - index koji ide od 0,1,2

duboko kopiranje
promjena se vidi, obje varijbale pokazuju na istu listu
lista1 = [0,1,2]
lista2 = lista1

plitko kopiranje
imena pokazuju na razlicite liste
lista1 = [0,1,2]
lista2 = lista1[:]

ako lista sadrzi podliste onda je potrebno dubokokopiranje
import copy
copy.**deepcopy**(x)

l = [0,1,2,3,4,5]
l[1:5] = [10]
print l
[0,10,5] --> na mjesta od 1 do 5 se kopira lista 10, da je bilo [4,4,4] onda bi se to ubacilo

N - torke

- nepromijenjiva, brze izvodenje
- index, count, len, in

ntorka = ('p','d',3,4)

n = tuple([1,2,3])
lista = list(n)
(x,y,z) = (1,3,5)

Skupovi --> set

- nema duplikata, razliciti tipovi
- s = {1, (2,3), '4', 5, 5}
- skup = {1, 2, 3, 4}
- skup.add(4)
- skup.add(5)
- skup.update({4,6})
- skup.update([4,6]) --> dodaj listu, nema razlike u odnosu na set

skup.discard(4) - uklanja element, nema exceptiona
skup.remove(4) - uklanja element, ima exceptiona ako nema elementa
skup.pop() - baca exception ako je prazan, pop uzima nasumicno odabran

Izbacivanje duplikata iz liste

- l = list(set(l))

Operacije nad skupovima

- union |
- intersection &
- differnece -
- jenakost ==
- issuperset nadskupa
- issubset podskup

Mape (Dictionary)

- jedinstveni kljuc, vrijednost ne mora biti
- mapa = {}
- mapa = dict ()
  - mapa = {'1000' : 1000}
  - v1 = mapa['1000'] --> exception ako nema
  - v2 = mapa.get('1000',-1) --> vrati -1 default vrijednost ako nema kljuca 1000
  - mapa['10']= 1001
  - del mapa['10'] --> pop je napredniji, pop(key, default), oboje dizu exception
  - mapa.keys(), mapa.values(), '1000' in mapa

Red

ili
l=[]
l.insert(0,1)
l.insert(0,2)
l.pop()
l.pop()

```
from collections import deque
q = deque()
q.append(1)
q.append(2)
```

- Napisati funkciju `dupli(l)` koja ispisuje broj duplih elemenata u listi `l`

```
def dupli(l):
    # create an empty dictionary to store element counts
    counts = {}

    # loop over each element in the list and count occurrences
    for elem in l:
        if elem in counts:
            counts[elem] += 1
        else:
            counts[elem] = 1

    # count number of elements with counts greater than 1
    num_duplicates = sum(1 for count in counts.values() if count > 1)

    # print the number of duplicate elements
    print("Number of duplicates:", num_duplicates)
```

```
CITIES = {
    'zg': 1,
    'st': 2
}


def add(cities: map, name: str, pop: int):
    cities[name] = pop
    return cities

def remove(cities : map, name : str):
    if name in cities:
        del cities[name]
        return cities

a = add(CITIES, "a", 1)
print(a)
b = remove(CITIES,"a1")
print(b)

print(max(CITIES.values()))
print(min(CITIES.values()))
```

String

- s.count(podrijec) --> broj pojavljivanja
- s.index(podrijec) --> index u string
- x,y,z = map(float, input().split())

```
def napravi_kratice(recenica):
    kratica = ""
    for rijec in recenica.split():
        if len(rijec) <= 2:
            continue
        kratica += rijec[0].upper()
    return kratica
```

```
def postotak_rijeci_nisu_veznici(recenica):
    rijeci = recenica.split()
    broj_rijeci = len(rijeci)
    broj_veznici = sum(rijec in {'i', 'pa', 'te', 'ni', 'niti', 'a', 'ali', 'nego', 'no',
'ili'} for rijec in rijeci)
    broj_rijeci_nisu_veznici = broj_rijeci - broj_veznici
    postotak = broj_rijeci_nisu_veznici / broj_rijeci * 100
    return postotak


def duljine_rijeci_nisu_veznici(recenica):
    rijeci = recenica.split()
    rijeci_nisu_veznici = [rijec for rijec in rijeci if rijec not in {'i', 'pa', 'te',
'ni', 'niti', 'a', 'ali', 'nego', 'no', 'ili'}]
    duljine = {rijec: len(rijec) for rijec in rijeci_nisu_veznici}
    return duljine


def remove_vowels(word):
    """Izbacuje sve samoglasnike iz riječi"""
    vowels = "aeiouAEIOU"
    return "".join([char for char in word if char not in vowels])

def swap_case(string):
    """Zamjenjuje velika i mala slova u stringu"""
    return string.swapcase()

def add_spaces(word):
    """Ubacuje razmak između svakog susjednog slova u riječi"""
    return " ".join(list(word))

def is_anagram(str1, str2):
    """Provjerava može li se drugi string dobiti premetanjem znakova
prvog stringa"""
    return sorted(str1) == sorted(str2)


def ispravi_rečenicu(rečenica):
    riječi = rečenica.split()
    ispravljene_riječi = []
    for riječ in riječi:
        if riječ:
            ispravljena_riječ = riječ.strip().capitalize()
            ispravljene_riječi.append(ispravljena_riječ)
    ispravljena_rečenica = ' '.join(ispravljene_riječi)
    if not ispravljena_rečenica.endswith('.'):
        ispravljena_rečenica += '.'
    return ispravljena_rečenica
```

```
recenica = 'Ovo je rečenica s nekim riječima koje
nisu veznici i neke koje su veznici.'

postotak =
postotak_rijeci_nisu_veznici(recenica)
print(f'Postotak riječi koje nisu veznici:
{postotak:.2f}%')

duljine = duljine_rijeci_nisu_veznici(recenica)
print('Duljine riječi koje nisu veznici:')
for rijec, duljina in duljine.items
```

For petlja

- range(start,end,step)

Komprehenzija

```
for i in range(10):
    nova.append('Number' + str(i))
```

```
nova = ['Number' + str(i) for i in range(10)]
```

Datoteke

- f = open(path, 'r|w')
- f.read() --> cijeli sadrzaj
- f.readline() --> redak
- f.readlines() --> retke kao listu stringova
- f.write() --> string u datoteku
- f.writelines() --> lista stringova kao retke
- f.close --> zatvara

```
with open(path) as f:
    for line in f:
        print(line)
```

Stdin/stdout

- python obrada.py < podaci.txt
- python obrada.py < podaci.txt > izlaz.txt

```
from collections import Counter
from typing import List, Tuple

def read_data(filename: str = None) -> Tuple[int, List[str], str]:
    # Ako nije navedeno ime datoteke, čitamo sa standardnog ulaza
    if filename is None:
        file = open(0) # 0 označava standardni ulaz
    else:
        file = open(filename, 'r')

    people = [line.strip().split(', ') for line in file]
    file.close()

    num_people = len(people)

    # Sortiramo osobe po prezimenu silazno
    sorted_people = sorted(people, key=lambda x: x[1], reverse=True)

    # Pronalazimo najčešće ime
    names = [person[0] for person in people]
    most_common_name = Counter(names).most_common(1)[0][0]

    return num_people, sorted_people, most_common_name
```

Doseg varijabli

```
def outer_function():
    x = "local"

    def inner_function():
        nonlocal x
        x = "nonlocal"
        print("Inner function:", x)

    inner_function()
    print("Outer function:", x)

outer_function()
```

stdout -->
Inner function: nonlocal
Outer function: nonlocal

Moduli

```
import moj                  from moj import *

moj.funkcija()              funkcija()
```

- paket, **direktorij** modula

```
import paket.modul
import scipy.stats
scipy.stats.variation(a)

from paket import modul
from scipy import stats
stats.variation(a)

from paket.modul import objekt
from scipy.stats import variation
variation(a)
```

Instalacije pakta
- pip install **ime_paketa**
- pip install **ime_paketa**==verzija
- pip install upgrade **ime_paketa**

provjera instaliranih
- pip freeze
- pip show **ime_paketa**

deinstalacija
- pip uninstall **ime_paketa**

```
Import putanje
from data_models.myClass import ImeKlase
```

```
# POMOCNI.PY
def median(x, y, z):
    nums = [x, y, z]
    nums.sort()
    return nums[1]


# GLAVNI.PY
from pomocni import median

a = 5
b = 10
c = 7

med = median(a, b, c)
print(f"Srednji broj od {a}, {b} i {c} je {med}.")
```

```
# GLAVNI.PY
from utils.pomocni import median

a = 5
b = 10
c = 7

med = median(a, b, c)
print(f"Srednji broj od {a}, {b} i {c} je {med}.")
```

```
# GLAVNI.PY
import sys
sys.path.append('/putanja/do/direktorija/sa/modulom')

from pomocni import median

a = 5
b = 10
c = 7

med = median(a, b, c)
print(f"Srednji broj od {a}, {b} i {c} je {med}.")
```

Klase

- **primjerak** klase --> objekt
- **podaci** unutar klase --> atributi

statički atributi i metode

```
class Zaposlenik
    brojac =  0
    def __init__ (self, ime):
        Zaposlenik.brojac +=1

    def izbroji():
        return Zaposlenik.brojac
```

```
class Zaposlenik:
    broj_zaposlenika = 0

    def __init__(self, ime, prezime, placa=5000):
        self.ime = ime
        self.prezime = prezime
        if placa is not None:
            self.placa = placa
        else:
            self.placa = None
        Zaposlenik.broj_zaposlenika += 1

    @staticmethod
    def izbroji():
        return Zaposlenik.broj_zaposlenika

    def __str__(self):
        if self.placa is not None:
            return f"{self.ime} {self.prezime}, placa: {self.placa}"
        else:
            return f"{self.ime} {self.prezime}"
```

```
# stvaranje objekata klase Zaposlenik
ana = Zaposlenik("Ana", "Anić", 6000)
ivan = Zaposlenik("Ivan", "Ivić")
maja = Zaposlenik("Maja", "Majić", None)

# ispis broja zaposlenika
print(Zaposlenik.izbroji())  # izlaz: 3

# ispis podataka o zaposlenicima
print(ana)  # izlaz: Ana Anić, placa: 6000
print(ivan)  # izlaz: Ivan Ivić
print(maja)  # izlaz: Maja Majić
```

Nasljedivanje

```
class Zaposlenik(Osoba) :
```

- zaposlenik nasljeduje i prosiruje od Osobe

```
class Zaposlenik:
    def __init__(self, ime, prezime, placa=0):
        self.ime = ime
        self.prezime = prezime
        if placa:
```

```
# primjer korištenja
ana = Zaposlenik("Ana", "Anić", 5000)
maja = Student("Maja", "Majić", 3000)
petar = Zaposlenik("Petar", "Perić", 4000)
poduzece = Poduzece("Moj Biznis", "Zagreb")
poduzece.dodaj_zaposlenika(ana)
poduzece.dodaj_zaposlenika(maja)
poduzece.dodaj_zaposlenika(petar)
```

```
    def __init__(self, ime, prezime, placa=0):
        self.ime = ime
        self.prezime = prezime
        if placa:
            self.placa = placa
        else:
            self.placa = None

    def __str__(self):
        return f"{self.ime} {self.prezime}, placa: {self.placa}"

    @staticmethod
    def izbroji(zaposlenici):
        return len(zaposlenici)


class Student(Zaposlenik):
    def __init__(self, ime, prezime, placa=0):
        super().__init__(ime, prezime, placa)

    def __str__(self):
        return f"{self.ime} {self.prezime} (student), placa: {self.placa}"


class Poduzece:
    def __init__(self, ime, lokacija):
        self.ime = ime
        self.lokacija = lokacija
        self.zaposlenici = []

    def dodaj_zaposlenika(self, zaposlenik):
        self.zaposlenici.append(zaposlenik)

    def daj_povisicu(self, postotak):
        for zaposlenik in self.zaposlenici:
            if zaposlenik.placa:
                zaposlenik.placa *= (1 + postotak/100)

    def ukupni_porez(self):
        ukupni_porez = 0
        for zaposlenik in self.zaposlenici:
            if zaposlenik.placa:
                if isinstance(zaposlenik, Student):
                    ukupni_porez += zaposlenik.placa * 0.1
                else:
                    ukupni_porez += zaposlenik.placa * 0.24
        return ukupni_porez

    def izbaci_zaposlenika(self, zaposlenik):
        self.zaposlenici.remove(zaposlenik)

    def __str__(self):
        output = f"{self.ime}, lokacija: {self.lokacija}\n"
        output += "Zaposlenici:\n"
        for zaposlenik in self.zaposlenici:
            output += f"- {str(zaposlenik)}\n"
        return output
```

Polimorfizam

- neku klasu koristimo kao i njezinu nadklasu (neki zaposlenik moze biti student ali svi dobivaju povisice)

- skrivanje atributa klase

Apstrakcija

- javne su samo one potrebne

```
class Zaposlenik:
    def __obavijest(self):
        pass
    def postavi_placu(self, vrijednost):
        self.__placa = vrijednost
```

```
poduzece = Poduzece("Moj Biznis", "Zagreb")
poduzece.dodaj_zaposlenika(ana)
poduzece.dodaj_zaposlenika(maja)
poduzece.dodaj_zaposlenika(petar)
poduzece.daj_povisicu(10)
poduzece.izbaci_zaposlenika(maja)
print(poduzece)
print(f"Ukupni porez: {poduzece.ukupni_porez()} kn")
```

Iterator

```
class Poduzece:
  def __init__(self):
    self.zaposlenici = []

  def dodaj_zaposlenika(self, zaposlenik):
    self.zaposlenici.append(zaposlenik)

  def __iter__(self):
    self.trenutni_zaposlenik = 0
    return self

  def __next__(self):
    if self.trenutni_zaposlenik < len(self.zaposlenici):
      trenutni = self.zaposlenici[self.trenutni_zaposlenik]
      self.trenutni_zaposlenik += 1
      return trenutni
    else:
      raise StopIteration

moje_poduzece = Poduzece()

zaposlenik1 = Zaposlenik("Ivo", "Ivić", "ivo@ivoc.com", "1234567890")
zaposlenik2 = Zaposlenik("Ana", "Anić", "ana@anic.com", "0987654321")

moje_poduzece.dodaj_zaposlenika(zaposlenik1)
moje_poduzece.dodaj_zaposlenika(zaposlenik2)

for zaposlenik in moje_poduzece:
  print(zaposlenik)
```

Izniminke

```
while True:
try:
    x = int(input("Unesite broj :"))
    break
except (RuntimeError , TypeError , NameError):
    pass
    # ignoriranje iznimke ne preporučuje se!

raise NameError ('HiThere')
```

Custom exception

```
class MyError(Exception):
  def __init__(self, value):
    self.value = value
  def __str__(self):
    return "MyError with value " + str(self.value)

try:
    raise MyError(5)
except MyError as e:
    print(e)
```

```
class ErrZap(Exception):
  pass

class ErrZapDodaj(ErrZap):
  def __init__(self, zaposlenik):
    self.zaposlenik = zaposlenik

  def __str__(self):
    return f"Zaposlenik {self.zaposlenik} već postoji u poduzeću."

class ErrZapIzbaci(ErrZap):
  def __init__(self, zaposlenik):
    self.zaposlenik = zaposlenik

  def __str__(self):
    return f"Zaposlenik {self.zaposlenik} ne postoji u poduzeću."
```

```
p = Poduzece()

try:
  p.dodaj_zaposlenika("Ana Anic")
  p.dodaj_zaposlenika("Ivo Ivic")
  p.dodaj_zaposlenika("Ana Anic")  # ova linija će podići iznimku ErrZapDodaj
except ErrZapDodaj as e:
  print(e)

try:
  p.izbaci_zaposlenika("Pero Peric")  # ova linija će podići iznimku ErrZapIzbaci
except ErrZapIzbaci as e:
  print(e)
```

```
class Poduzece:
   def __init__(self):
      self.zaposlenici = []

   def dodaj_zaposlenika(self, zaposlenik):
      if zaposlenik in self.zaposlenici:
         raise ErrZapDodaj(zaposlenik)
      else:
         self.zaposlenici.append(zaposlenik)

   def izbaci_zaposlenika(self, zaposlenik):
      if zaposlenik not in self.zaposlenici:
         raise ErrZapIzbaci(zaposlenik)
      else:
         self.zaposlenici.remove(zaposlenik)
```

Serijalizacija

```
# Serijalizacija objekta u datoteku
p = Poduzece()
p.dodaj_zaposlenika("Ana Anic")
p.dodaj_zaposlenika("Ivo Ivic")
with open("poduzece.pickle", "wb") as f:
   pickle.dump(p, f)
```

```
import pickle

# Učitavanje serijaliziranog objekta iz datoteke
with open("poduzece.pickle", "rb") as f:
   p = pickle.load(f)

# Ispisivanje zaposlenika iz učitanog objekta
for zaposlenik in p.zaposlenici:
   print(zaposlenik)
```

Os

```
subprocess.run(["firefox", "skripta.pdf"])

os.getcwd --> pwd
os.chdir(dirname) --> cd
os.listdir --> dir
os.rename(oldPath, newPath)
os.path.basename(path) --> ime datoteke
os.path.splittext(path) --> ('/home/dat','.txt')
```

```
import os

# Definiranje putanje direktorija
dir_path = "/path/to/directory"

# Pronalazak abecedno prve datoteke
prva_datoteka = sorted(os.listdir(dir_path))[0]

# Stvaranje poddirektorija tmp
os.mkdir(os.path.join(dir_path, "tmp"))

# Stvaranje nove putanje za datoteku
nova_putanja = os.path.join(dir_path, "tmp", "prva" +
os.path.splitext(prva_datoteka)[1])

# Premještanje datoteke u tmp direktorij i mijenjanje imena
stara_putanja = os.path.join(dir_path, prva_datoteka)
os.rename(stara_putanja, nova_putanja)
```

HTTP

```
query = {'lat ': '45', lon ': '45'}
r = requests. get('http://api.open',
params=query)
print(r. headers)
print( r. json)

files = {'file':open('report.xls','rb')}
r= requests.post(url, files=files)

"files":{
      "file":"<censored>"
}
```

```
import requests

# Definiranje URL-a za API zahtjev
url = "http://api.open-notify.org/astros.json"

# Dohvaćanje podataka o astronautima pomoću GET zahtjeva
response = requests.get(url)

# Pretvaranje odgovora u format JSON
response_json = response.json()

# Ispis imena astronauta koji se trenutno nalaze u ISS
print("Astronauti u ISS-u:")
for astronaut in response_json["people"]:
   print(astronaut["name"])
```

6

Wednesday, April 5, 2023 12:10 AM

Proces
- program koji se izvrsava

Dretve
- izvrsavanje unutar istog procesa

GIL
- istodobni pristup py intrepreteru

Kada koristiti threadove
- responsive UI
- task delgetaion

```
import threading
import time
# Kod koji se izvršava u neovisnoj dretvi
def countdown(n):
while n > 0:
print('t-minus', n)
n -= 1
time.sleep(1)
# Stvori i pokreni dretvu
t = threading.Thread(target=countdown, args=(10,))
t.start() # eksplicitni početak
```

Upravljanje stanjem dretve

```
# provjeri je li dretva živa
t.is_alive()
# blokiraj trenutnu dretvu dok se dretva t ne završi
t.join()
```

Kritični odsječci (za dijeljeni pristup)

```
lock = threading.Lock()
lock.acquire()
lock.release()
    with lock:
# kritični odsječak..
```

Queue

instanciranje --> red = queue.Queue()
ubacivanje --> red.put(value)
cekanje --> value= red.get()

Primjer

```
import threading
import queue
work_queue = queue.Queue()
# Stvaranje dretvi:
NUM_THREADS = 4
threads = [
threading.Thread(target=worker, args=(work_queue,))
for i in range(NUM_THREADS)
]
# Pokreni dretve
for thread in threads:
thread.start()


#Funkcija koju izvodi pojedina dretva
def worker(work_queue):
while True:
item = work_queue.get()
# ... obradi item i ispisi rezultat ...
work_queue.task_done()

#Glavna dretva koja ubacuje podatke
for i in items:
work_queue.put(i)
```

```
import threading
import queue

# Funkcija koju dretve izvršavaju
def worker():
   while True:
      item = work_queue.get()
      if item is None:
         # Ako je element None, to znači da su svi elementi obrađeni
         # i dretva završava s radom
         break
      # Ovdje se obavlja obrada elementa
      # ...
      work_queue.task_done()

# Inicijalizacija reda za radne zadatke
work_queue = queue.Queue()

# Dodavanje elemenata u red
# ...

# Pokretanje dretvi
num_threads = 4
threads = []
for i in range(num_threads):
   t = threading.Thread(target=worker)
```

```
#Glavna dretva koja ubacuje podatke
for i in items:
work_queue.put(i)

# Čišćenje:
# čekaj dok se ne dobiju i ne obrade svi elementi reda
work_queue.join()

# čekaj da sve dretve završe
while threads:
threads.pop().join()
```

Stvori još jedan red:

```
results_queue = queue.Queue()
```

Kreiraj bazen dretvi koje kao argumente primaju oba reda:

```
Thread(target=worker, args=(work_queue, results_queue))
```

U worker funkciji, dodaj rezultat u red za rezultate:

```
results_queue.put(rezultat obrade itema)
```

Ispiši sve rezultate:

```
while not results_queue.empty():
print(results_queue.get()
```

Novi proces fork()

```
import os
child_pid = os.fork()
if child_pid == 0:
print('Child Process: PID', os.getpid())
else:
print('Parent Process: PID', os.getpid())
```

Napredni fork

```
from multiprocessing import Process
def f(name):
    print('hello', name)
if __name__ == '__main__':
    p = Process(target=f, args=('bob',))
    p.start() # pokretanje
    p.join() # čeka završetak

from multiprocessing import Process
import os
def info(title):
    print(title)
    print('module name:', __name__)
    print('parent process:', os.getppid()) # roditeljev ID
    print('process id:', os.getpid()) # moj ID
def f(name):
    info('function f')
    print('hello', name)
if __name__ == '__main__':
    info('main line')
    p = Process(target=f, args=('bob',))
    p.start()
    p.join()
```

Socket

Stvaramo ie modulom socket standardne biblioteke:

```
num_threads = 4
threads = []
for i in range(num_threads):
  t = threading.Thread(target=worker)
  t.start()
  threads.append(t)

# Označavanje kraj reda s None elementima
for i in range(num_threads):
  work_queue.put(None)

# Čekanje da sve dretve završe
for t in threads:
  t.join()

# Svi elementi su obrađeni, program se nastavlja
```

Ispisati HTML sadržaj jedne od web stranica Google i Bing, one koja prva odgovori na zahtjev.

- Svaka od dvije dretve paralelno dohvaća jedan URL i rezultat sprema u red
- Glavna dretva ispisuje prvi element iz reda rezultata čim se on pojavi (red.get())
- Dohvaćanje HTML-a u worker funkciji dretve:

```
import urllib.request
sadrzaj = urllib.request.urlopen(url).read()
```

- Dohvaćeni HTML treba ispisati u datoteku
- Na kraju pozvati naredbu (web browser) koja će ga otvorit

```
import threading
import urllib.request
import queue
import webbrowser

# Funkcija koju izvršavaju dretve
def worker(url, result_queue):
  try:
    # Dohvaćanje HTML sadržaja
    with urllib.request.urlopen(url) as response:
      html_content = response.read()

    # Spremanje HTML sadržaja u red rezultata
    result_queue.put(html_content)
  except Exception as e:
    print(f'Error fetching URL: {url}, {e}')

# Lista URL-ova za dohvaćanje
urls = ['https://www.google.com', 'https://www.bing.com']

# Red za spremanje rezultata
result_queue = queue.Queue()

# Pokretanje dretvi
threads = []
for url in urls:
  t = threading.Thread(target=worker, args=(url, result_queue))
  threads.append(t)
  t.start()

# Čekanje na završetak svih dretvi
```

Socket

Stvaramo je modulom socket standardne biblioteke:
s = socket.socket(address family, socket type)

Za IPv4 adrese upotrebljavamo socket.
AF_INET

Za TCP protokol upotrebljavamo
socket.SOCK_STREAM

Povezujemo je na određeno sučelje metodom:
s.bind((HOST, PORT))

Omogućujemo joj prihvat konekcija metodom:
s.listen()

Na klijentskoj strani:
s.connect((SERVER_HOST, SERVER_PORT)

28/33

```
    t.start()

# Čekanje na završetak svih dretvi
for t in threads:
    t.join()

# Ispis prvog rezultata iz reda
if not result_queue.empty():
    html_content = result_queue.get()
    print('HTML content:')
    print(html_content)

    # Spremanje HTML sadržaja u datoteku
    with open('web_page.html', 'wb') as f:
        f.write(html_content)

    # Otvaranje HTML sadržaja u web pregledniku
    webbrowser.open('web_page.html')
else:
    print('No results in queue.')
```

Kreirajte dva procesa: jedan računa zbroj svih vrijednosti sin(x) za x = 1, 2, ..., 107
, a drugi zbroj svih cos(x) za x = 1, 2, ..., 107
. Glavni
proces treba ispisati rezultat koji prije završi

```
import multiprocessing
import math

# Function to calculate sum of sin(x)
def calculate_sin_sum(start, end):
    sin_sum = 0
    for x in range(start, end + 1):
        sin_sum += math.sin(x)
    return sin_sum

# Function to calculate sum of cos(x)
def calculate_cos_sum(start, end):
    cos_sum = 0
    for x in range(start, end + 1):
        cos_sum += math.cos(x)
    return cos_sum

if __name__ == '__main__':
    # Set the range for x values
    start = 1
    end = 10**7

    # Create two processes
    sin_process = multiprocessing.Process(target=calculate_sin_sum,
args=(start, end))
    cos_process = multiprocessing.Process(target=calculate_cos_sum,
args=(start, end))

    # Start the processes
    sin_process.start()
    cos_process.start()

    # Wait for the processes to finish
    sin_process.join()
```

```
cos_process.join()

# Print the results
sin_sum = sin_process.exitcode
cos_sum = cos_process.exitcode

print("Sum of sin(x):", sin_sum)
print("Sum of cos(x):", cos_sum)

# Print the result of the process that finishes first
if sin_process.exitcode < cos_process.exitcode:
   print("Sum of sin(x) finished first.")
else:
   print("Sum of cos(x) finished first.")
```

MI 2021

Saturday, April 22, 2023 11:10 AM

ispit je podijeljen u 2 dijela, 18 bodova su pitanja na zaokruživanje, a 17 je pisanje programskog koda, prošle godine je bilo napisati klasu pravokutnik i onda da klasa kvadrat naslijedi tu klasu ili tak nesto
uglavnom taj programski mislim da nece bit teži od 1. labosa