Danmarks Tekniske Universitet

Kursusnavn	Introduktion til programmering og databehandling
Kursusnummer	02631 (02632, 02633, 02634, 02692)
Hjælpemidler	Alle hjælpemidler er tilladt
Varighed	2 timer
Vægtning	Opgaverne vægtes ens

Indhold

Aflevering

Du skal aflevere dine løsninger elektronisk:

1. Du kan uploade dine løsninger individuelt på CodeJudge

   https://dtu.codejudge.net/prog-e20/assignments

   under *Exam*. Når du afleverer en løsning på CodeJudge, bliver det test-eksempel, som fremgår i opgavebeskrivelsen, kørt på din løsning. Hvis din løsning består denne ene test, vil den fremgå som *Submitted*. Det betyder, at din løsning består på dette ene test-eksempel. Du kan uploade til CodeJudge så mange gange som du ønsker under eksamen.

2. Du skal uploade alle dine løsninger på DTU's Onlineeksamen site. Hver assignment skal uploades som en separat .py fil med samme navn som funktionen i opgaven:

   (a) `robust_mean.py`
   (b) `shape_roundness.py`
   (c) `candy_exchange.py`
   (d) `change_case.py`
   (e) `nearest_shop.py`

   Filerne skal afleveres separat (*ikke* som én zip-fil) og skal have præcis disse filnavne.

Efter eksamen vil dine løsninger fra DTU Inside blive automatisk evalueret på CodeJudge på en række forskellige tests, for at undersøge om de generelt fungerer korrekt. Bedømmelsen af din aflevering foretages udelukkende på baggrund af, hvor mange af de automatiserede test der bestås.

- Du skal sikre dig, at din kode følger specifikationerne nøje.
- Hver løsning må ikke indeholde nogen yderligere kode udover den specificerede funktion, dog kan `import`-sætninger inkluderes.
- Husk at du kan tjekke om dine løsninger følger specifikationerne ved at uploade dem til CodeJudge.
- Bemærk at alle vektorer og matricer anvendt som input eller output skal være numpy arrays.

Assignment 1 Robust mean

Middelværdien af værdier x_n hvor $n = 1, \ldots, N$ kan blive påvirket af outliers (ekstreme observationer). For at se bort fra outliers, kan du udregne en robust midelværdi, *robust mean*, som følgende. Først, udregn midelværdien μ og standardafvigelsen σ som

$$\mu = \frac{1}{N}\sum_{n=1}^{N} x_n\,, \quad \sigma = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_n - \mu)^2}\,.$$

Betragt nu værdier x_n som ikke afviger mere end σ fra μ, og fjern andre værdier (outliers). Det vil sige, du skal beholde værdier hvor

$$\mu - \sigma \leq x_n \leq \mu + \sigma.$$

Lad os sige at værdier, som du beholder, er x_k, $k = 1, \ldots, K$. Den robuste middelværdi r udregnes nu som middelværdien af disse værdier

$$r = \frac{1}{K}\sum_{k=1}^{K} x_k\,.$$

Problemdefinition

Skriv en funktion `robust_mean` som tager en vektor med nogle værdier som input, og returnerer den robuste middelværdi af værdierne.

Løsningsskabelon

```
def robust_mean(x):
    # insert your code
    return r
```

Input	
`x`	En vektor med værdier x_n.

Output	
`r`	En skalar med den robuste middelværdi af x_n.

Eksempel

Betragt værdierne

$$\mathbf{x} = [\,5.4\,,\quad 17.4\,,\quad 5.5\,,\quad 6.4\,,\quad 4.3\,]\,.$$

Middelværdien og standardafvigelsen er

$$\mu = \frac{1}{5}(5.4 + 17.4 + 5.5 + 6.4 + 4.3) = 7.8$$

$$\sigma = \sqrt{\frac{1}{5}\left((5.4 - 7.8)^2 + (17.4 - 7.8)^2 + (5.5 - 7.8)^2 + (6.4 - 7.8)^2 + (4.3 - 7.8)^2\right)} = 4.846\,.$$

Vi har $\mu - \sigma = 2.9540$ og $\mu + \sigma = 12.6460$. Kun en værdi, værdien 17.4, skal fjernes som en outlier, da værdien er for stor. Den robuste middelværdi er

$$r = \frac{1}{4}(5.4 + 5.5 + 6.4 + 4.3) = \underline{5.4}\,.$$

Assignment 2 Shape roundness

En form kan repræsenteres med en boolsk (*Boolean, logical*) matrix $\mathbf{S}$ hvor `True` (`1`) repræsenterer formen og `False` (`0`) repræsenterer baggrund. Du kan forstille dig, at dette er et mønstre af kvadratiske sort-og-hvide fliser eller billedpixels.

Et mål for formens rundhed er forholdet mellem formens areal A og formens omkreds P

$$r = \frac{A}{P}\,.$$

Her er arealet A summen af arealer af alle kvadraterne i formen, mens omkredsen P er summen af længder af alle kanterne, der afgrænser formen.

Problemdefinition

Skriv en funktion `shape_roundness` som tager en boolsk matrix, der repræsenterer formen, som input, og returnerer målet for formens rundhed.

Løsningsskabelon

```
def shape_roundness(S):
    # insert your code
    return r
```

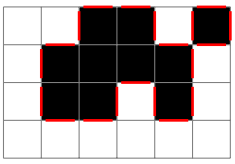
Input	
`S`	En boolsk matrix (elementerne er enten `True` (`1`) eller `False` (`0`)) som repræsenterer formen.

Output	
`r`	Et decimaltal med formens rundhed.

Eksempel

Betragt matricen $\mathbf{S}$ (vi skriver 0 for `False` og 1 for `True`) og formen som den repræsenterer.

$$\mathbf{S} = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



Formens areal måles ved at tælle de sorte kvadrater. For matricen $\mathbf{S}$ er arealet $A = 10$. Formens omkreds måles ved at tælle de lodrette og vandrette røde kanter, der afgrænser formen. For matricen $\mathbf{S}$ er omkredsen $P = 20$.

Formens rundhed er

$$r = \frac{10}{20} = \underline{0.5}\,.$$

2

Assignment 3 Candy exchange

Ejeren af en slikbutik har en plan for at komme af med slikpapir smidt på vejen foran butikken: "Aflever fem stykker slikpapir, og få et stykke slik gratis!" Dette fører til spørgsmålet: Hvis du har et bestemt antal stykker slik, hvor mange stykker slik kan du faktisk ende med at spise? Her antager vi, at du bliver ved med at aflevere slikpapir så længe som muligt.

Problemdefinition

Skriv en funktion `candy_exchange` som tager antallet af stykker slik du har som input, og returnerer antallet af stykker slik, du kan ende med at spise.

Løsningsskabelon

```
def candy_exchange(n):
    # insert your code
    return m
```

Input	
`n`	Et heltal med antallet af stykker slik som du har.

Output	
`m`	Et heltal med antallet af stykker slik som du kan ende med at spise.

Eksempel

Lad os sige, at du starter med $n = 73$ stykker slik.

Efter du har spist slikket, sidder du tilbage med 73 stykker slikpapir. Din status er: 73 spiste stykker slik og 73 stykker slikpapir.

Du bytter 70 stykker slikpapir til 14 nye stykker slik, som du så spiser. Din status er: 87 spiste stykker slik (73 fra før og 14 nye) og 17 stykker slikpapir (3 som du ikke kunne bytte og 14 nye).

Du bytter 15 stykker slikpapir til 3 nye stykker slik, som du så spiser. Din status er: 90 spiste stykker slik (87 fra før og 3 nye) og 5 stykker slikpapir (2 som du ikke kunne bytte og 3 nye).

Du bytter 5 stykker slikpapir til et nyt stykke slik, som du så spiser. Din status er: 91 spiste stykker slik (90 fra før og 1 nyt) og 1 stykke slikpapir, som du ikke kan bytte til flere slik.

Status er opsummeret nedenfor.

candies eaten	73	87	90	91
wrappers	73	17	5	1

Du kan ende med at spise

$$m = \underline{91}$$

stykker slik.

3

Assignment 4 Change case

To konventioner for navngivning af variable som består af flere ord er: *snake case* (slangeskrift) og *camel case* (kamelskrift). *Snake case* bruger en understreg i stedet for et mellemrum. *Camel case* angiver starten på et nyt ord med et enkelt blokbogstav. Et eksempel på *snake case* er `circle_radius`, og tilsvarende *camel case* er `circleRadius`. Variablenavne med et ord er ens i de to konventioner.

Problemdefinition

Skriv en funktion `change_case` som tager en streng med variablenavn i enten *snake case* eller *camel case*, og returnerer en streng med variablenavn i den anden konvention. Variablenavn kan bestå af et eller flere ord.

Løsningsskabelon

```
def change_case(v):
    # insert your code
    return u
```

Input

`v`	En streng med variablenavnet i enten *snake case* eller *camel case*. Variablenavn kan bestå af et eller flere ord.

Output

`u`	En streng med variablenavnet i den anden konvention end den brugt for input streng.

Eksempel

Betragt variablenavnet

`v = 'thisIsIdentifierName'`.

Blokbogstaverne i navnet afslører at dette er *camel case*. Det fulde navn kan deles i individuelle ord

this is identifier name

Skrevet i *snake case* variablenavnet er

`u = 'this_is_identifier_name'`.

Assignment 5 Nearest shop

En butikskæde bruger postnumre for at henvise kunder til den nærmeste butik. Til dette har de en liste med 4-cifrede postnumre for deres butikker. Når kunden indtaster et 4-cifret postnummer, bliver postnummeret på den nærmeste butik fundet som følgende:

- Hvis kundens postnummer er ens med et af butikkens postnumrer, er dette den nærmeste butik.
- Hvis der ikke findes en butik med et postnummer, som helt matcher kundens postnummer, betragtes butikker, hvor de tre første cifre i postnumret er ens med de tre første cifre i kundens postnummer. Hvis der findes sådanne 3-cifrede matches, er det match, som står først på listen, den nærmeste butik.
- Hvis der ikke findes 3-cifrede matches, betragtes 2-cifrede matches på en lignende måde. Hvis der ikke findes nogle 2-cifrede matches, betragtes 1-cifrede matches.
- Hvis ingen butik matcher det første ciffer i kundens postnummer, er butikken der vises først på listen, den nærmeste butik.

Problemdefinition

Opret en funktion `nearest_shop` der tager et 4-cifret heltal og en vektor, der indeholder 4-cifrede heltal. Funktionen skal returnere det element fra vektoren, der identificerer den nærmeste butik.

Løsningsskabelon

```
def nearest_shop(c, s):
    # insert your code
    return n
```

Input	
`c`	Et 4-cifret heltal som representerer kundens postnummer.
`s`	En vektor med 4-cifrede heltal som repræsenterer butikkens postnumre.

Output	
`n`	Et 4-cifret heltal med postnummeret på den nærmeste butik.

Eksempel

Betragt kundens postnummer `c` = 3490 og en liste med butikkens postnumre

$$\texttt{s} = [\,2300\,,\quad 1890\,,\quad 3990\,,\quad 3590\,,\quad 7900\,]\,.$$

På listen med butikkens postnumre findes kundens postnummer 3490 ikke. Der er ingen butiksnumre der starter med 349, og heller ingen butiksnumre, der starter med 34. Der er to postnummre, der starter med 3: nummer 3990 og nummer 3590. Af disse to numre skal den, der står først på listen, returneres, og det er nummeret

$$\texttt{n} = \underline{3990}\,.$$