Bevezetés a Python használatába

Aradi Bernadett

2017/18 ősz

Kurzusinformációk:

`https://arato.inf.unideb.hu/aradi.bernadett`

A kurzushoz tartozó jegyzet:

`https://gyires.inf.unideb.hu/GyBITT/19/Neuralis_halozatok_v8.pdf`

```
import this
```

A Python adattípusai

- Számok: integer és float
- boolean: `True`, `False`
- string: `print('Hello World!')`

Az adattípusok dinamikusan (deklarálás nélkül) adódnak:

- `type(1)`
- `type(1.2)`
- `type(1.)`
- `type("Hello")` # string: ' vagy "
- `type(None)`
- `type(int)`

Változók típusának konvertálása:

- `str(17)`
- `int("17")`
- `float("2.5")`
- `float("1")`

Alapműveletek

`x=3`

- `print(x)`
- `print(x + 2), +,-,*,/`
- `print(x ** 2)` # hatványozás
- `x += 1`
  `print(x)`
- `x *= 5`
  `print(x)`
- `7 // 3` # hányados egészrésze ⇒ `type: int`
- `7 % 3` # osztási maradék

Adhatunk értéket egyszerre több változónak:

- `x,y=2,8`
- `print(x*y)`

Indentálás Pythonban!!

```
x=3
if x>1:
  x+=2
print(x)
```

Kommentelés:

- 1-soros: #
- többsoros: """ előtte-utána
- Alt használatával: egyszerre több sor kijelölése (jupyter notebook)

Változónevek:

- a–z, A–Z, 0–9, _
- nem kezdődhetnek számmal
- case sensitive
- nem lehetnek az alábbiak:

```
and   assert   break   class   continue   def   del
elif   else   except   exec   finally   for   from
global   if   import   in   is   lambda   not   or
pass   print   raise   return   try   while   yield
```

Logikai operátorok

```
t=True
f=False
```

- `print(type(t))`
- `print(t and f)`
- `print(t or f)`
- `print(not t)`
- `print(t != f)` # XOR

A `bool` típus interpretálható `int`-ként is: `True==1` és `False==0`

- `1 == 1`
- `3 * 2 == 7`
- `2 != 2`
- `3 * 2 != 7`
- `1 < 5`
- `3 >= 0`
- `1 < 2 >= 3`

Műveletek stringekkel

- `x = "Hello, I'm"`
  `x + "Python!"` `# Default: utf-8 karakterkódolás`
- `h = 'Hello'`
  `w = "world"`
  `print(h)`
- `print(len(h))` `# string hossza`
- `hw = h + ' ' + w + '!'` `# stringek összefűzése`
  `print(hw)`
- `"Do you have {no} of those {what}?".format(no='two',what='apples')`
- `'Do you have %d of those %s' % (2,'apples')`

`s="hello"`

- indexelés: `s[0]`, `s[1]`, ..., `s[5]`
- negatív indexelés: `s[-1]`, `s[-2]`, ..., `s[-5]`
- `s.capitalize()`, `s.upper()`, `s.rjust(7)`, `s.center(9)`
- `s.replace('l', 'k')`
- `' world '.strip()` `# eltünteti a szóközöket előtte-utána`

A Python objektumai

- listák: akár különböző típusú változók rögzített sorrendben
- szótárak (dictionary): (key,value) párokat rögzít
- halmazok
- osztályok (class): definiálhatunk egyéb típusokat
- függvények

Listák

- `lst = [3, 1, 2]`
- `print(lst,lst[2])`
- `print(lst[-1])`
- `lst[0]='start'`
  `print(lst)`
- `lst.append(3)`
- `ls=ls.pop()` # kiveszi a lista utolsó elemét
  `print(ls,lst)`
- `empty=[]`
- `lst1=[0,1,2,3]`
  `lst2=[4,5,6]`
  `lst=lst1+lst2` # listák összefűzése
  `print(lst)`
- `nums=list(range(7))`
  `print(nums)` # range: beépített fv egész számok sorozatára

'Slicing lists'

- `nums=list(range(7))`
- `print(nums)`
- `print(nums[2:4])` # slice: 2 indexű elemtől 3-ig
- `print(nums[2:])`
- `print(nums[:4])`
- `print(nums[:])` # teljes lista
- `print(nums[:-1])` # az utolsó elemig
- `print(nums[1:5:2])` # ez hogy jött ki?
- `print(nums[4::-2])` # visszafelé is mehetünk
- `print(nums[::-1])`

↑ stringekre ugyanígy!

- `nums[2:4]=[100,101]`
  `print(nums)`
- `nums[2:4]=list(range(20))`
  `print(nums)`

Listák...

- ```
  nums=list(range(7))
  letters=['a','b','c']
  lst=[nums,letters]
  print(lst[1][2])
  print(lst[1][:2])
  ```
- `len(nums)`, `len(lst)`
- `len([])`
- `len('python')`
- `8 in nums` # in: ellenőrzi, hogy az elem benne van-e a listában
- `'t' in 'python'`
- `min(nums)`, `max(nums)`
  ```

Listák rendezése

```
lst = [25, 50, 10, -150, 28, 250, 310, 33]
```

- `sorted(lst)` # növekvő sorrend
- `sorted(lst, reverse=True)` # csökkenő sorrend

Vektorok, mint szabályos sorozatok

A `range` függvény segítségével:

- `list(range(4))`
- `list(range(4,10))`
- `list(range(2, 12, 3))`
- `list(range(-7, -37, -5))`

for-ciklus egy lista elemei mentén

- ```
  hallgatok=['Sanyi','Anna','Gábor']
  for valaki in hallgatok:
    print(valaki, 'jár neurális gyakorlatra')
  ```
- ```
  hallgatok=['Sanyi','Anna','Gábor']
  for idx, valaki in enumerate(hallgatok):    # enumerate
    print(valaki, 'a névsor', idx+1,'.  tagja')
  ```
- ```
  hallgatok=['Sanyi','Anna','Gábor']
  for idx, valaki in enumerate(hallgatok):
    print('%s a névsor %d.  tagja' % (valaki,idx + 1))
  ```
- ```
  nums = [0, 1, 2, 3, 4]
  negyzet = []
  for i in nums:
    negyzet.append(i ** 2)
  print(negyzet)
  ```

Függvények

- Függvények definiálása: `def` parancs segítségével:
- ```
  def sign(x):
    if x > 0:
      return 'positive'
    elif x < 0:
      return 'negative'
    else:
      return 'zero'
  ```
- ```
  for x in [-1, 0, 1]:
    print(sign(x))
  ```

Feladatok

Állítsuk elő minél egyszerűbben az alábbi vektorokat!

1. $a = (0, 1, \ldots, 30)$
2. $b = (2, 4, 6, \ldots, 100)$
3. $c = (2, 1.9, 1.8, \ldots, 0)$
4. $d = (0, 3, 6, \ldots, 27, 30, -100, 30, 27, \ldots, 6, 3, 0)$

3. feladat megoldása:

```
c = list(range(20,-1,-1))
c = [i/10 for i in c]
print(c)
```

Feladatok 2

Legyen v egy adott 100 elemű vektor. Ennek megadására egy lehetőség:

```
from random import randint
v=[randint(-500,500) for i in range(0,100)]
print(v)
```

5. Állítsuk elő azt a w vektort, melynek elemei a v elemei fordított sorrendben felsorolva!

Írjunk függvényt, amely

6. egy adott n természetes szám esetén kiírja $n!$ értékét;
7. egy adott természetes szám esetén eldönti, hogy az palindrom szám-e!

NumPy

Tömbök kezeléséhez, numerikus számításokhoz készült kiegészítő csomag a Pythonhoz.

```
import numpy as np
```

- alapvető adatszerkezete az n-dimenziós tömb
- vektor és mátrix szintű utasítások
- lineáris algebrai és véletlenszám almodul
- több magasabb szintű modul is épül rá
  (pl. scipy, matplotlib, scikit-learn)

Tömbök létrehozása konkrét adatokból

Vektorok:

- `v = np.array([1, 3, 2])`
- `print(type(v))`
- `print(v.shape)`
- `v[0]=5`
- `print(v)`

Mátrixok:

- `A = np.array([[1,2,3],[4,5,6]])`
  `# sorfolytonosan mátrixot képez`
- `print(A.shape)`
- `A[0,1]=-100`
- `print(A)`

`import numpy as np` helyett:

- `from numpy import array`
- `v = array([1, 3, 2])`

Speciális tömbök

- `B = np.zeros((3,2))` # csupa 0-ból álló mátrix
- `C = np.ones((1,2))` # csupa egyesből álló mátrix
- `D = np.full((2,5),7)` # konstans mátrix
- `E = np.eye(3)` # 3x3-as egységmátrix
- `F = np.random.random((2,2))`
  # véletlen számok $[0, 1]$-en egyenletes eloszlásból

Indexelés, slicing:

- mint a listáknál. DE:
- `A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])`
- `B = A[:2, 1:3]`
- `print(B)`
- `B[0, 0] = 1000`
- `print(A)`

Még több slicing: amire figyelni kell...

- `A = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])`

a középső sor kiválasztása:

- `sor1 = A[1, :]` vagy `sor2 = A[1:2, :]` `# ugyanaz?`
- `print(sor1)`
- `print(sor2)`
- `print(sor1,sor1.shape)`
- `print(sor2,sor2.shape)`

`sor1 = A[1, :]` – integer indexing: csökkenti a tömb rangját
`sor2 = A[1:2, :]` – slicing: nem csökkenti a rangot

Integer array indexing

- `B = np.array([[1,2], [3, 4], [5, 6]])`
- `print(B[[0, 1, 2], [0, 1, 0]])`
- `print(np.array([B[0, 0], B[1, 1], B[2, 0]]))`
- `print(B[[0, 0], [1, 1]])`
  `# tehát többször is választhatjuk ugyanazt az elemet`
- `C = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])`
- `v = np.array([0, 2, 0, 1])`
- `print(C[np.arange(4), v])` `# minden sorból egy elem`
  `# arange`: NumPy range függvénye tömbökre
- `C[np.arange(4),v] += 100`
  `# minden sorból egy elemhez hozzáadunk 100-at`
- `print(C)`

Boolean array indexing

Egy logikai feltétel ellenőrzésével történik az indexelés.

- `C = np.array([[1,2], [3, 4], [5, 6]])`
- `felt = (C > 1)` # visszatérési érték: boolean NumPy tömb
- `print(felt,felt.shape)`
- `print(C[felt])` # 1 rangú NumPy tömb ⇒ vektor
- `print(C[C>1])` # közvetlenül

Adattípusok NumPy-ban:

Mint korábban, itt is dinamikusan választ adattípust a Python. Ennek ellenére egy opcionális argumentummal ki is kényszeríthetünk bizonyos adattípus-választást:

- `v = np.array([1, 2])`
  `print(v.dtype)` # típus: `int32`
- `v = np.array([1., 2])`
  `print(v.dtype)` # típus: `float64`
- `v = np.array([1, 2],dtype=np.float32)`
  `print(v.dtype)` # típus: `float32`

Műveletek NumPy tömbökkel

A tömbökkel végzett matematikai alapműveletek elemenként hajtódnak végre, és függvényként is meghívhatjuk őket:

- `A = np.array([[1,2],[3,4]])`
  `B = np.array([[5,6],[7,8]])`
- `print(A+B)` # mátrixösszeadás
  `print(np.add(A,B))`
- `print(A-B)` # különbségképzés
  `print(np.subtract(A,B))`
- `print(A*B)` # szorzás elemenként!
  `print(np.multiply(A,B))`
- `print(A/B)` # osztás
  `print(np.divide(A,B))`

Egyéb matematikai függvények is elemenként:

- `print(np.sqrt(A))`

Mit kapunk, ha 0-val osztunk, vagy negatív számból vonunk négyzetgyököt?

Belső/skaláris szorzat, mátrixszorzás

A `dot` függvénnyel történik:

- ```
  A = np.array([[1,2],[3,4]])
  B = np.array([[5,6],[7,8]])
  v = np.array([9,10])
  w = np.array([11, 12])
  ```
- ```
  print(np.dot(v, w))      # v és w vektorok belső szorzata
  print(v.dot(w))      # ez is
  ```
- ```
  print(np.dot(A,v))      # A mátrix és v vektor szorzata
  print(A.dot(v))      # ez is
  print(np.dot(v,A))      # és ez?
  ```
- ```
  print(np.dot(A,B))      # mátrixszorzás
  print(A.dot(B))      # ez is
  ```

Vegyük észre, hogy a NumPy nem tesz különbséget sor- és oszlopvektorok között!

Ha mégis sor- és oszlopvektorokkal szeretnénk dolgozni: $n \times 1$ és $1 \times n$ típusú mátrixokra van szükségünk:

- `v = np.array([9,10])`
  `print(v==v.T)` `# .T: transzponálás`
- `v = np.array([[9,10]])` `# v sorvektor`
  `w = np.array([[9],[10]])` `# v oszlopvektor`
  `print(v.shape,w.shape) print(v==w)`
  `print(v==w.T)`

NumPy függvények

- `A = np.array([[1,2],[3,4]])`
  `print(np.sum(A))` `# elemek összege`
- `print(np.sum(A, axis=0))` `# összeg oszloponként`
- `print(np.sum(A, axis=1))` `# összeg soronként`

Egyéb hasznos NumPy függvények:
`https://docs.scipy.org/doc/numpy/reference/routines.math.html`

Feladatok 3

1. Állítsuk elő azt a 3×4-es mátrixot, mely sorfolytonosan tartalmazza az $1,\ldots,12$ elemek kétszereseit!
   (Hint: használjuk az `np.reshape` függvényt!)
2. Állítsuk elő a csupa 1-esekből álló 3×5-ös mátrixot.
3. Állítsuk elő azt az 5×3-as mátrixot, melynek átlójában az 1,2,3 elemek állnak, minden más eleme 0.
4. Keressük meg egy 6×8-as véletlen mátrix minden sorában a maximális elemet.
   (Hint: a mátrix megadásához használjuk ismét az `np.reshape` függvényt, valamint például a `randint` parancsot!)
5. Írjunk függvényt, amely egy adott n természetes szám esetén (for-ciklus használata nélkül) kiírja $n!$ értékét!

Broadcasting

Különböző méretű NumPy tömbökkel végzett műveletvégzés esetén hasznos.
Például tegyük fel, hogy egy rögzített vektort hozzá szeretnénk adni egy mátrix minden sorához:

```
A = np.array([[1,2],[3,4],[5,6]])
v = np.array([2,1])
```

Megoldás:

- for-ciklussal
- broadcasting segítségével: `print(A+v)`

$\rightsquigarrow$ elemenkénti műveletvégzés
az ezt támogató függvények: univerzális függvények, lásd:
`https://docs.scipy.org/doc/numpy/reference/ufuncs.html`

Mit tegyünk, ha a

```
w = np.array([1,1,3])
```

vektort szeretnénk hozzáadni `A` minden oszlopához?

Broadcasting – hogyan működik?

1. Ha a 2 tömb rangja nem egyezik meg: a kisebb rangú tömb shape-vektorának elejére annyi 1-est ír, hogy a rangok megegyezzenek.
   Pl. (3,1,4) és (2,) $\rightsquigarrow$ (3,1,4) és (1,1,2).
2. Két tömböt egy adott dimenzióban kompatibilisnek mondunk, ha a két tömb adott dimenziója megegyezik, vagy az egyik tömb esetén ez az érték 1.
   Pl. (3,1,4) és (1,1,2) az első 2 dimenziójukban kompatibilisek.
3. Két tömb akkor "broadcastolható", ha minden dimenziójukban kompatibilisek.
   Pl. (3,1,4) és (1,1,2) nem "broadcastolható".
4. Két tömb "broadcastolása" esetén mindkét tömb úgy viselkedik, mintha minden dimezióban a 2 lehetőség maximuma lenne az adott dimenzió mérete.
   Pl. (3,1,1) és (1,2,4) esetén: (3,2,4)
5. Ha valamelyik dimenzióban a két tömb mérete különböző, akkor az 1 méretűt annyiszor egymás után másolja, hogy a két tömbbel elemenként végezhessük a műveleteket.

Függvényábrázolás

Matplotlib: beépített könyvtár ábrák készítéséhez, ezen belül a matplotlib.pyplot egy függvényábrázolásra használható modul.

```
import matplotlib.pyplot as plt
```

Pl. a cos függvény ábrázolása egy tartományon:

- `x = np.arange(0,3*np.pi,0.1)` # pontok x koordinátája
- `y = np.cos(x)` # megfelelő y értékek kiszámolása
- `plt.plot(x, y)`
- `plt.show()` # ez a parancs teszi láthatóvá a grafikát

Több függvény ábrázolása és feliratok készítése:

- `y2 = np.sin(x)`
- `plt.plot(x, y,'b')`
- `plt.plot(x, y2,'g:')`
- `plt.xlabel('x-tengely')`
- `plt.ylabel('y-tengely')`
- `plt.title('cos és sin függvények')`
- `plt.legend(['cos', 'sin'])` # jelmagyarázat készítése
- `plt.show()`

Feladatok 4

1. Olvassuk be egy tömbbe a
   `https://arato.inf.unideb.hu/baran.agnes/NH2017/elemek.txt`
   címen található adatállományt!
   Erre egy lehetőség:
   ```
   import urllib.request
   url='...'      # ide kell a fenti link
   points = np.loadtxt(urllib.request.urlopen(url))
   ```
2. Vizsgáljuk meg az adathalmazunk felépítését!
   1-esekkel és -1-esekkel felcímkézett síkbeli pontok szerepelnek benne.
   Hány 1-es és hány -1-es jelzésű pontunk van?
3. Ábrázoljuk egy közös koordináta-rendszerben különböző színnel az 1-es, valamint a -1-es címkével rendelkező pontokat.

Perceptronnal ketté fogjuk választani a 2 ponthalmazt. Előtte:

4. Tegyük rá az ábránkra a -2 meredekségű, 10-es tengelymetszettel rendelkező egyenest is egy harmadik színnel.

A perceptron algoritmus

1. Vizsgáljuk meg, hogyan függ a szükséges korrekciós lépések száma a tanulási paramétertől!
2. Mi történik, ha közelebb hozzuk egymáshoz a két halmazt? Pl. legyen $B = B + \varepsilon$, ahol $\varepsilon > 0$.
3. Ellenőrizzük, hogy teljesül-e a perceptron konvergencia tétel!
4. Ábrázoljuk grafikonon, hogy hogyan alakul a hiba az epochok függvényében!