Școala
informală
de IT

Python Development

Pythonic Data Structures

Cuprins

Liste

Pythonic Data Structures

Liste

- Lista este o structură de date ordonată și mutable (poate fi modificată).
- Permite elemente duplicat.
- Este definită folosind parantezele drepte [ ].

```
my_list = [8, 2, -3, 6, 20]
```

- În Python, o listă poate conține orice tip de date. Cu toate acestea, de cele mai multe ori listele o să aibă același tip de date astfel încât conținutul lor să fie intuitiv.

```
my_list = [4+5j, 'Ana are mere', -20.75, True, None]
```

- Listele sunt structuri de date ale căror elemente pot fi accesate folosindu-ne de un index. Fiecare element are un index (o poziție) care-l definește. Numerotarea acestora începe de la **0** și se termină la **n-1** (unde n reprezintă numărul de elemente din listă.
- Pentru accesarea unui element ne vom folosi de numele listei precedat de index-ul dorit scris între paranteze drepte [ ].

```
my_list = [4+5j, 'Ana are mere', -20.75, True, None]
print(my_list[0])  # (4+5j)
print(my_list[1])  # Ana are mere
print(my_list[2])  # -20.75
print(my_list[3])  # True
print(my_list[4])  # None
```

Liste

- Elementele din listă pot fi accesate și de la final spre început prin folosirea indexilor negativi, astfel ultimul element va fi identificat prin indexul **-1**, iar primul element va fi identificat prin indexul **-n** (unde n reprezintă numărul de elemente din listă).

```
my_list = [4+5j, 'Ana are mere', -20.75, True, None]
print(my_list[-1])  # None
print(my_list[-2])  # True
print(my_list[-3])  # -20.75
print(my_list[-4])  # Ana are mere
print(my_list[-5])  # (4+5j)
```

- Listele sunt o structură de date cu un număr finit (definit) de elemente. Pentru a afla lungimea (numărul de elemente) unei liste avem la dispoziție funcția **len()**.

```
my_list = [4+5j, 'Ana are mere', -20.75, True, None]
print(len(my_list))  # 5
```

Liste

- Un alt mod de accesare a elementelor unei liste o reprezintă conceptul de slice (împărțire). Cu ajutorul acestei noțiuni vom obține o nouă listă pe baza unei liste inițiale.
- Operația de slice are mai multe forme:
  - putem specifica doar index-ul de start. Acesta poate fi definit folosind index pozitiv (de la începutul listei) sau negativ (de la coada listei)

```
# Specificam elementul de start folosind index pozitiv.
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[4:]
print(my_sliced_list)  # [4, 5, 6, 7, 8, 9, 10]
```

```
# Specificam elementul de start folosind index negativ.
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[-5:]
print(my_sliced_list)  # [6, 7, 8, 9, 10]
```

  - putem specifica doar index-ul de final. Acesta poate fi definit folosind index pozitiv sau index negativ.

```
# Folosind index pozitiv.
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[:6]
print(my_sliced_list)  # [0, 1, 2, 3, 4, 5]
```

```
# Folosind index negativ.
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[:-5]
print(my_sliced_list)  # [0, 1, 2, 3, 4, 5]
```

Liste

- putem specifica atât indexul de start cât și cel de final. Atenție, elementul de pe indexul de final NU va fi inclus în lista rezultată

```
# Folosind index pozitiv.
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[4:7]
print(my_sliced_list)  # [4, 5, 6]
```

```
# Folosind index negativ.
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[-5:-2]
print(my_sliced_list)  # [6, 7, 8]
```

- cazul de mai sus poate fi folosit și prin combinarea indexilor (primul pozitiv, al doilea negativ sau vice-versa)

```
# Folosind combinatie de indexi - pozitiv:negativ
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[2:-5]
print(my_sliced_list)  # [2, 3, 5, 6]
```

```
# Folosind combinatie de index - negativ:pozitiv
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[-5:8]
print(my_sliced_list)  # [6, 7]
```

- un alt mod de a folosi metoda de slice este definirea pasului. Acesta definește din câte în câte elemente să fie formată noua listă pe baza indexilor de start și final.

```
# Folosind toata lista si pas.
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[::2]
print(my_sliced_list)  # [0, 2, 4, 6, 8]
```

```
# Folosind doar index de start si pas
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[1::2]
print(my_sliced_list)  # [1, 3, 5, 7, 9]
```

```
# Folosind doar index de final si pas
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[:7:3]
print(my_sliced_list)  # [0, 3, 6]
```

```
# Folosind index de start,final si pas
my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
my_sliced_list = my_list[2:-3:4]
print(my_sliced_list)  # [2, 6]
```

Liste

- Listele dispun de o varietate de metode cu ajutorul cărora putem obține diferite funcționalități:
  - **.index(element)** - returnează indexul elementului ***element*** (prima poziție unde este găsit)
  - **.append(element_nou)** - adaugă un elementul ***element_nou*** la finalul listei.
  - **.insert(index, element_nou)** - adaugă elementul ***element_nou*** pe poziția ***index***.
  - **.remove(element)** - scoate din listă elementul ***element***.
  - **.pop()** - scoate din listă ultimul element.
  - **.pop(index)** - scoate din listă elementul de pe poziția ***index***.
  - **.clear()** - scoate din listă toate elementele.
  - **.copy()** - copiază lista într-o altă listă. Mai poate fi folosită și funcția list(): **list(my_list)**.
  - **.reverse()** - construiește lista având elementele în ordine inversă (de la final la început).
  - **.sort()** - sortează elementele din listă în ordine ascendentă (modificarea se face in-place).
  - **.count(element)** - returnează de câte ori se află elementul ***element*** în listă.
  - concatenarea a două liste se face folosind operatorul **+**: ***list_3 = list_1 + list_2***. Prin această metodă vom obține o altă listă, iar fiecare listă inițială își va păstra datele.
  - concatenarea a două liste se poate folosind funcția ***extend()***: **list_1.extend(list_2)**. Prin această metodă elementele din ***list_2*** vor fi adăugate la finalul listei ***list_1***.

Tupluri

Pythonic Data Structures

Tupluri

- Tuplul este o structură de date ordonată și immutable (nu poate fi modificată).
- Permite elemente duplicat.
- Este definit folosind parantezele rotunde ( ).

```
my_tuple = (1, 7, -3, 'a', 2+3j, True, None, False)
```

- Fiind o structură de date ordonată permite aceleași modalități de sliceing ca listele (inclusiv concatenare, folosind operatorul +).
- Fiind o structură de date immutable nu permite operații de modificare: adăugare sau ștergere.
- Dacă avem nevoie să folosim o colecție de date ordonată a cărei structură nu se va modifica (immutable) este indicat să folosim tuplurile în detrimentul listelor. Deși la prima vedere îndeplinesc aceleași roluri și putem folosi o listă în loc de tuplu, tuplurile ocupă mai puțină memorie.

```
print(().__sizeof__())  # 24 biti pentru tuplu
print([].__sizeof__())  # 40 biti pentru lista
```

- Un exemplu pentru folosirea tuplului este definirea lunilor din an. Vor fi aceleași 12, deci nu este nevoie de modificarea lor.

```
months_of_year = ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec')
```

Dicționare

Pythonic Data Structures

Dicționare

- Dicționarul este o structură de date neordonată, mutable (poate fi schimbată) și indexată.
- Datele dintr-un dicționar sunt reprezentate sub forma de **cheie:valoare**.
- Este definit folosit acolade { }.
- Într-un dicționar cheile sunt unice, valorile pot fi duplicat.

```
my_dictionary = {
    "key_1": 12,
    "key_2": 4+5j,
    3: True,
    4: None,
    5+2j: '',
    ("key_6", 7): [1, 2, 3],
    -8: ('First', 'Second', 'Third')
}
```

- Din câte se poate observa din exemplul anterior, cheia poate fi string, număr sau chiar tuplu.

Dicționare

- O valoarea dintr-un dicționar poate fi accesată în mai multe feluri:
  - prin accesarea directă a cheii. În acest caz vom avea o eroare dacă cheia nu există în dicționar.

```
my_dict = {
    "key_1": "My value"
}
print(my_dict["key_1"])  # va afisa "My value"
```

  - folosind metoda **.get()**. Avantajul acestei metode este că putem defini o valoare default dacă cheia nu există. Această valoare o specificăm ca al doilea parametru al funcției. Dacă acest parametru va fi omis și cheia nu există în dicționar, metoda va întoarce **None**.

```
my_dict = {
    "key_1": "My value"
}
print(my_dict.get("key_1"))  # va afisa "My value"
print(my_dict.get("key_2"))  # va afisa None (default)
print(my_dict.get("key_1", "My custom value"))  # va afisa "My value"
print(my_dict.get("key_2", "My custom value"))  # va afisa "My custom value"
```

Dicționare

- Are disponibile următoarele metode:
  - **clear()** - golește dicționarul.
  - **copy()** - returnează o copie a dicționarului.
  - **keys()** - returnează un **dict_keys** conținând toate cheile dintr-un dicționar.
  - **values()** - returnează un **dict_values** conținând toate valorile dintr-un dicționar.
  - **items()** - returnează un **dict_items** conținând toate elementele dintr-un dicționar. Un astfel de element reprezintă un tuplu de forma ***(cheie, valoare)***.
  - **pop(key)** - scoate din dicționar cheia ***key***.
  - **popitem()** - scoate din dicționar ultimul element introdus.

Seturi

Pythonic Data Structures

Seturi

- Setul este o colecție de date neordonată, immutable (nu poate fi modificat) și neindexată.
- Este definit folosind acolade **{ }**.
- Elementele dintr-un set sunt unice.

```
my_set = {"item_1", 2, 3+4j, True, False}
```

- Accesarea unui element din set nu este posibilă, dar setul poate fi parsat.
- Pentru eliminarea unui element din set se pot folosi metodele **remove()** sau **discard()**. Poate fi folosită și metoda **pop()**, dar această funcție va scoate din set ultimul element - setul fiind neordonat nu vom avea control asupra acestui element.
- Metoda **clear()** este folosită pentru golirea setului.
- Pentru concatenarea a două seturi poate fi folosită metoda **union()** - aceasta returnează un set nou - sau **update()** - aceasta adaugă valorile dintr-un set în altul (in-place update),
- Alte metode uzuale sunt **intersection()**, **issubset()**, **issuperset()**.

Git & GitHub

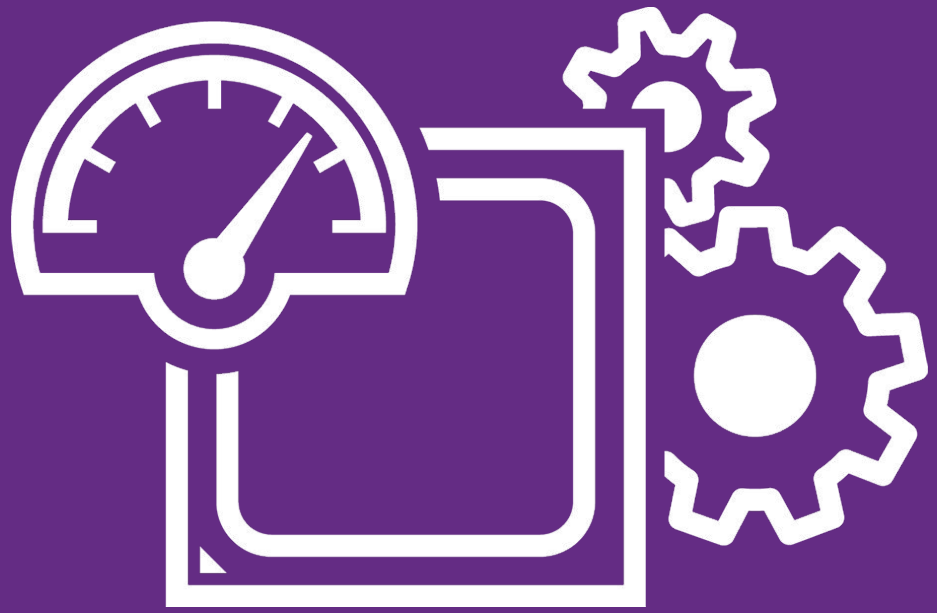
Pythonic Data Structures

Git & GitHub

- **Git** este un sistem distribuit de versionare ce are rolul să gestioneze un proiect de orice dimensiune cu viteză și eficiență sporită.
- Funcționalitatea ce face Git-ul să depășească în popularitate aproape orice alt SCM (Source Code Management) existent este **modelul de branching**.
    - alte SCM-uri existente: Subversion, CVS, Perforce, ClearCase ș.a.
- Git-ul permite folosirea multiplelor branch-uri ce pot fi independente.
- Din moment ce cam toate operațiile sunt făcute local, oferă un mare avantaj din punct de vedere al vitezei.
- **GitHub**-ul este o platformă ce permite găzduirea unui manager de versiuni și colaborarea între developeri.

Git & GitHub

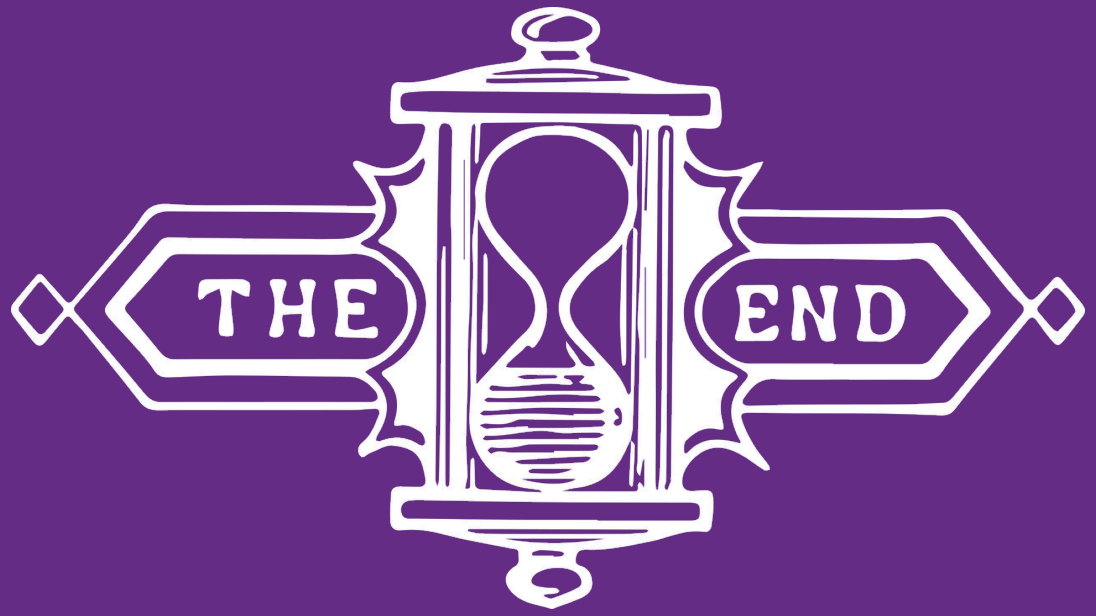
- Terminologie:
  - **repository** - în teorie este o colecție de fișiere și istoria schimbărilor suferite de acestea; în practică reprezintă proiectul nostru și toate fișierele aferente acestuia.
  - **clone** - este o comandă cu ajutorul căreia putem clona local un repository.
  - **branch** - reprezintă o versiune a proiectului care diverge din proiectul principal.
  - **checkout** - reprezintă schimbarea branch-urilor (mutarea de pe un branch pe altul).
  - **add** - acțiunea de a adăuga fișierele într-o zona intermediara numită Staging Area.
  - **commit** - acțiunea de a înregistra modificările făcute într-un repository.
  - **push** - acțiunea de a uploada versiunea locală pe server.
  - **merge** - reprezintă procesul de îmbinare a două versiuni diferite.
  - **pull** - acțiunea de a obține de la server ultimele modificări ale datelor.
  - **pull request** - reprezintă procesul de code review și merge al unui branch.



Temă

Temă

- Creați un script Python care îndeplinește următoarele funcții:
    - declară o listă care conține elementele 7, 8, 9, 2, 3, 1, 4, 10, 5, 6 (în această ordine).
    - afișează lista inițială ordonată ascendent (lista inițială trebuie păstrată în aceeași formă)
    - afișează lista inițială ordonată descendent (lista inițială trebuie păstrată în aceeași formă)
    - afișează o listă ce conține numerele pare din lista ordonată (folosind slice)
    - afișează o listă ce conține numerele impare din lista ordonată (folosind slice)
    - afisează o listă ce conține numerele ce sunt multipli ai numărului 3 (folosind slice).



Vă mulțumesc!