Școala
informală
de IT

Google

Python Development

Week 2. Pythonic “Hello, World!”

Cuprins

1. Program “Hello, World!”
2. Sintaxă de bază
3. Numere
4. Operatori
5. Variabile
6. Șiruri de caractere
7. Liste
8. Tupluri
9. Dicționare
10. Seturi

Program “Hello, World!”

Week 2. Pythonic “Hello, World!”

Program “Hello, World!”

- Programul **“Hello, World!”** este cel mai simplu program din lume.
- Este un program minimal al cărui rol este să exemplifice sintaxa și modul de funcționare al unui limbaj.
- Este folosit în procesul de învățare al unui limbaj de programare nou.
- Rezultatul programului este afișarea mesajului **Hello, World!** pe ecranul unui calculator.
- Pentru a scrie acest program, deschideți interpretorul Python și scrieți următoarea instrucțiune:

```
print("Hello, World!")
```

- Rezultatul este uimitor! În terminal apare mesajul **Hello, World!** ca prin magie.
- Din acest moment puteți dezvolta orice program doriți folosind Python. Tot ce trebuie să faceți este să schimbați mesajul pe care să-l afișați.

Program “Hello, World!”

- Folosind interpretorul Python în mod interactiv, avem dezavantajul de a nu putea refolosi codul.
- Pentru a dezvolta mai ușor programe Python și pentru a putea refolosi codul trebuie să creăm un script.
- Un script Python este un fișier cu extensia **.py** ce poate fi rulat de sine stătător. Acest fișier reprezintă programul nostru.
- Pentru scrierea codului Python poate fi folosit orice editor de texte. Pentru o viață mai ușoară, un mediu de dezvoltare mai rapid și mai performant vom folosi IDE-ul PyCharm.
- După crearea fișierului .py acesta poate fi rulat folosindu-vă de interpretorul Python în Script Mode. Folosiți interpretorul Python urmat de numele scriptului creat.
- Acesta va produce un rezultat similar cu executarea codului în mod interactiv.

Sintaxă de bază

Week 2. Pythonic “Hello, World!”

Sintaxă de bază

- Linia **print(“Hello, World!”)** reprezintă o instrucțiune Python.
- Cele mai importante reguli de sintaxă în programarea Python sunt:
  - o linie din program poate conține o singură instrucțiune. Instrucțiunile nu se termină cu nici un caracter sau grup de caractere. Pot exista mai mult instrucțiuni pe aceeași linie atât timp cât nici una din ele nu necesită un bloc de instrucțiuni. În acest caz acestea vor fi separate de caracterul ;.
  - blocurile de cod sunt obținute prin indentare (nu se folosesc nici un fel de paranteze sau alte caractere).
  - o linie de comentariu începe cu semnul #. Un comentariu poate fi adăugat la finalul unei instrucțiuni, pe aceeași linie cu aceasta. Comentariile nu sunt cod recunoscut și rulat de interpretor.

Numere

Week 2. Pythonic “Hello, World!”

Numere

- Numerele reprezintă un **TIP DE DATĂ**.
- În Python există trei tipuri de numere:
  - întregi (**int**) - numere întregi, fără virgulă, negative sau pozitive de lungime nelimitată:
    - 3, 5, 20, -17
  - raționale/zecimale (**float**) - numere cu virgulă, negative sau pozitive care conțin una sau mai multe zecimale (cifre după virgulă). Pot fi folosite valori științifice unde **e** reprezintă puteri ale lui 10.
    - 2.4, 3.0, -7.65, 8.46315, 5e2, -4e3, 1e-1
  - complexe (**complex**) - numere complexe care sunt reprezentate folosind litera **j** pentru definirea părții imaginare:
    - 1j, 3+5j, -8j
- Oricare din tipurile de numere de mai sus poate fi convertit la un alt tip folosind funcțiile **int()**, **float()**, respectiv **complex()**. Numerele complexe nu pot fi convertite în alt tip de numere.
- Tipul unei date/valori poate fi aflat folosind metoda **type()**.

Operatori

Week 2. Pythonic “Hello, World!”

Operatori aritmetici

Operator	Nume	Detalii	Exemplu	Rezultat
+	adunare	Adunarea matematică a două sau mai multe valori.	5 + 2	7
-	scădere	Scăderea matematică a două sau mai multe valori.	5 - 7	-2
*	înmulțire	Înmulțirea matematică a două sau mai multe valori.	3 * 4	12
/	împărțire	Împărțirea matematică a două sau mai multe valori.	5 / 2	2.5
%	modulo	Restul împărțirii unei valori la o altă valoare	7 % 4	3
**	ridicare la putere	Ridicarea unei valori la o anumită putere.	2 ** 4	16
//	împărțire exactă	Câtul împărțirii a două valori.	7 // 4	1

Operatori de comparare

- Orice comparație făcută va avea ca rezultat o dată de tip **Boolean**:
  - **True** - dacă rezultatul comparației are valoare de adevăr.
  - **False** - dacă rezultatul comparației nu are valoare de adevăr.

Operator	Nume	Descriere	Exemplu
==	egalitate	Este evaluată ca True dacă valoarea din stânga este egală cu valoarea din dreapta, altfel este evaluată ca False.	5 == 5
!=	inegalitate	Este evaluată ca True dacă valoarea din stânga este diferită față de valoarea din dreapta, altfel este evaluată ca False.	7 != 5
>	mai mare	Este evaluată ca True dacă valoarea din stânga este mai mare decât valoarea din dreapta, altfel este evaluată ca False.	5 > 3
<	mai mic	Este evaluată ca True dacă valoarea din stânga este mai mică decât valoarea din dreapta, altfel este evaluată ca False.	6 < 8
>=	mai mare sau egal	Este evaluată ca True dacă valoarea din stânga este mai mare sau egală cu valoarea din dreapta, altfel este evaluată ca False.	7 >= 7
<=	mai mic sau egal	Este evaluată ca True dacă valoarea din stânga este mai mică sau egală decât valoarea din dreapta, altfel este evaluată ca True.	4 <= 6

Operatori logici

- Operatorii logici sunt folosiți pentru combinarea rezultatelor obținute în urma condițiilor.

Operator	Nume	Descriere	Exemplu
and	ȘI LOGIC	Este evaluată ca True dacă ambele condiții sunt îndeplinite, altfel este evaluată ca False.	4 < 5 and 4 > 2
or	SAU LOGIC	Este evaluată ca True dacă cel puțin o condiție este adevarată, altfel este evaluată ca False.	4 < 6 or 8 < 5
not	NEGARE	Este evaluată ca True dacă valoarea din stânga este mai mare decât valoarea din dreapta, altfel este evaluată ca False.	not(4 < 5 and 4 > 2)

Operatori de identitate

- Operatorii de identitate sunt folosiți pentru compararea obiectelor.
- Nu se compară dacă obiectele sunt identice ci dacă sunt aceleași obiecte, adică dacă ocupă aceeași locație de memorie.

Operator	Descriere	Exemplu
is	Este evaluată ca True dacă ambele valori reprezintă aceeași locație de memorie, altfel este evaluată ca False.	7 is 7
is not	Este evaluată ca True dacă cele două valori reprezintă locații de memorie diferite, altfel este evaluată ca False.	7 is not 9

Operatori de apartenență

- Operatorii de apartenență sunt folosiți pentru verificarea existenței unei secvențe în cadrul unui obiect.

Operator	Descriere	Exemplu
in	Este evaluată ca True dacă secvența evaluată face parte din obiect, altfel este evaluată ca False.	1 in [1, 2, 3]
not in	Este evaluată ca True dacă secvența evaluată nu face parte din obiect, altfel este evaluată ca False.	‘a’ not in ‘python’

Operatori pe biți

- Informația într-un calculator este reprezentată în baza 2.
- Asta înseamnă ca orice informație este reprezentată printr-o secvență de biți ce pot avea fie valoarea 0, fie valoarea 1.

Operator	Nume	Descriere
&	AND	Setează fiecare bit cu valoarea 1 dacă ambii biți au valoarea 1, altfel setează valoarea 0.
|	OR	Setează fiecare bit cu valoarea 1 dacă cel puțin unul din biți este 1, altfel setează valoarea 0.
^	XOR	Setează fiecare bit cu 1 dacă doar unul din biți este 1, altfel setează valoarea 0.
~	NOT	Schimbă valoarea tuturor biților.
<<	left shift	Adaugă biți cu valoarea 0 la finalul secvenței.
>>	right shift	Elimină biții de la finalul secvenței.

Variabile

Week 2. Pythonic “Hello, World!”

Variabile

- O variabilă reprezintă o locație în memoria calculatorului ce poate fi identificată prin nume și conține informații (**valoare**).
- În Python nu există nici o instrucțiune pentru declararea unei variabile. O variabilă este creată în momentul în care aceasta este asignată.
- Alocarea și dealocarea memoriei se face de către Garbage Collector-ul Python. Spre deosebire de alte limbaje de programare, developerii Python nu își mai concentrează timpul și resursele pe această sarcină.
- Avantajul folosirii variabilelor îl reprezintă etichetarea unor valori și posibilitatea de refolosire a acestora.
- Numirea variabilelor reprezintă una din cele mai grele sarcini din repertoriul unui developer. Numele unei variabile trebuie să fie sugestiv atât pentru cine scrie codul cât mai ales pentru ceilalți din echipă. Numele unei variabile este dat pentru noi, developerii, nu pentru calculator.

Variabile - Naming convention

- Numele unei variable trebuie să fie intuitiv.
- În Python se folosește notarea **snake_case**.
- Numele acestora trebuie să înceapă cu litera mica.
- Numele variabilelor nu poate începe cu o cifră.
- Numele unei variabile poate să conțină orice caracter dintre a-z, A-Z, 0-9 și semnul _ (underscore).
- Python este un limbaj case-sensitive - există diferențe între variabila **age**, **Age** și **AGE**.

```
#Legal variable names:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"

#Illegal variable names:
2myvar = "John"
my-var = "John"
my var = "John"
```

Variabile - Asignare

- Pentru declararea unei variabile, în Python, este suficientă asignarea acesteia.
- Denumirea unei variabile se face în același moment în care aceasta se asignează.
- Pentru asignarea unei variabile vom folosi operatorul de asignare =

```
a = 3
```

- În exemplul anterior am asignat valoarea 3 unei variabile pe care am denumit-o a.
- În următorul exemplu vom da variabilei un nume mai intuitiv și vom afișa valoarea returnată de funcția **id()**.

```
child_age = 3

print(id(child_age))
```

- În Python, fiecare obiect este identificat printr-un ID unic. Acest ID este asignat obiectului în momentul în care acesta este creat și reprezintă adresa locației de memorie unde acesta este stocat.
- Acest ID este unic pentru fiecare rularea a programului, exceptând cazurile în care obiectele au asignat un ID unic constant, cum este cazul numerelor întregi de la -5 la 256.

Variabile - Asignare

- Asignarea unei variabile se poate face simplu, folosind operatorul =, ca în exemplul anterior sau prin orice combinație între operatorii aritmetici sau pe biți studiați anterior precedați de operatorul =.

```
# Asignare simpla: my_var va avea valoarea 5
my_var = 5

# Asignare prin adaugare: my_var va avea valoarea 8 (5 + 3)
my_var += 3  # echivalent cu my_var = my_var + 3

# Asignare prin inmultire: my_var va avea valoara 16 (8 * 2)
my_var *= 2  # echivalent cu my_var = my_var * 2

# Asignare prin shiftare catre dreaptea: my_var va avea valoarea 1 (16 >> 4 = 2)
# 0b10000 [16] >> 4 = 0b1 [1]
my_var >>= 4  # echivalent cu my_var = my_var >> 4

# Asignare prin shiftare catre stanga: my_var va avea valoarea 8 (1 << 3 = 8)
# 0b1 [1] << 3 = 0b1000 [8]
my_var <<= 3  # echivalent cu my_var = my_var << 3
```

Șiruri de caractere

Week 2. Pythonic “Hello, World!”

Șiruri de caractere

- Dacă în capitolele anterioare am discutat despre numere, acum a venit rândul caracterelor.
- Un șir de caractere reprezintă un alt tip de date alcătuit din unul sau mai multe caractere.
- Acest tip de date o să îl întâlniți sub denumirea de **string** (*pl. strings*).
- Un string este definit între ‘ ‘ (ghilimele simple) sau “ “ (ghilimele duble).
- Am folosit un string chiar în programul dezvoltat la începutul cursului:

```
print("Hello, World!")
```

- În instrucțiunea de mai sus am folosit funcția **print()** pentru a afișa mesajul **“Hello, World!”** (string-ul / șirul de caractere).
- Așa cum am folosit variabile cărora le-am asignat ca valori niște numere, la fel putem asigna și șiruri de caractere:

```
my_message = "Ana are mere."
print(my_message)
```

- În Python nu există noțiunea de caracter, un caracter poate fi reprezentat ca fiind un șir de caractere format dintr-un singur caracter.

```
my_char = "a"
```

- La fel de ușor poate fi definit și un șir de caractere gol:

```
my_empty_string = ""
```

Șiruri de caractere - ASCII

- Un șir de caractere poate conține orice caracter din codul ASCII.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

- Orice caracter din codul ASCII poate fi reprezentat printr-un număr întreg.
- Pentru a obține codul ASCII al unui caracter putem folosi funcția **ord()**.

```
print(ord('a'))
```

- Pentru a obține caracterul reprezentat de un cod ASCII putem folosi funcția **chr()**. `print(chr(97))`

Șiruri de caractere - escaping

- Există anumite caractere și/sau combinații de caractere care au un rol special. Cel mai simplu exemplu sunt ghilimelele - acestea (simple sau duble) sunt folosite pentru definirea unui șir de caractere.
- Pentru folosirea caracterelor speciale cu rol de caractere integrate în string-ul nostru vom folosi caracterul \ (backslash) pentru a face escape.

```
print("Si Dumnezeu a zis: \"Sa se faca lumina!\"")
```

- În exemplu anterior am folosit caracterul \ înaintea caracterului “ pentru a îl integra în șirul nostru de caractere, astfel încât mesajul final să fie: ***Si Dumnezeu a zis: “Sa se faca lumina!”***.
- O listă cu caracterele speciale o găsiți în imaginea următoare:

Escape Sequence	Meaning
newline	Ignored
\\	Backslash (\)
\'	Single quote (')
\"	Double quote (")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\v	ASCII Vertical Tab (VT)
ooo	ASCII character with octal value *ooo*
\x*hh...*	ASCII character with hex value *hh...*

Șiruri de caractere - multiline string

- Pentru scrierea pe mai multe linii există două variante:
  - folosirea caracterului ***newline***: \n

```
print("Somnoroase pasarele \nPe la cuiburi se aduna,\nSe ascund in ramurele-\nNoapte buna!")
```

  - folosirea ghilimelelor triple (simple sau duble):

```
print("""Somnoroase pasarele
Pe la cuiburi se aduna
Se ascund in ramurele -
Noapte buna!""")
```

- Cele două exemple de mai sus sunt echivalente, ambele vor afișa mesajul:

```
Somnoroase pasarele
Pe la cuiburi se aduna
Se ascund in ramurele -
Noapte buna!
```

Șiruri de caractere - raw string

- Pentru a evita rolul caracterelor speciale putem folosi noțiunea de **raw string**.

```
print('Somnoroase pasarele\nPe la cuiburi se aduna.')
```

- În exemplu anterior mesajul nostru va fi afișat pe două linii datorită caracterului newline \n.
- Folosind raw string putem afișa mesajul așa cum arată, fără a fi nevoiți să facem escape caracterului \n.
- Pentru definirea unui raw string vom prefixa string-ul nostru cu caracterul r sau R.

```
print(r'Somnoroase pasarele\nPe la cuiburi se aduna.')
```

- În acest exemplu mesajul va fi afișat exact așa cum apare între ghilimele. Această tehnică ne ajută când vrem să afișăm un text așa cum este, fără a îl formata.

Șiruri de caractere - formatting

- Un șir de caractere poate fi formatat prin mai multe metode. Prin formatarea unui string înțelegem înlocuirea unor placeholdere din cadrul string-ului cu date:
  - metoda **format()** - definirea placeholderului se face între acolade **{ }**

```
print("For only {price:.2f} dollars! {cheers_msg}".format(price=49, cheers_msg="Have a great day!"))
```

    - în exemplul anterior mesajul afișat va fi: ***For only 49 dollars! Have a great day!***
    - placeholderele pot fi identificate folosit nume (cum este și cazul de mai sus ***{cheers_msg}***), placeholdere numerotate sau chiar placeholdere goale:

```
print("For only {0:.2f} dollars! {1}".format(49, "Have a great day!"))
print("For only {:.2f} dollars! {}".format(49, "Have a great day!"))
```

    - toate exemplele de mai sus vor produce același rezultat.
    - în cazul ultimelor două cazuri trebuie respectată ordinea parametrilor primiți de funcția **format()**.
  - folosind F-strings - acestea au fost introduse pentru a face mai ușoară formatarea string-urilor:
    - nu mai este nevoie de definirea placeholderelor - formatarea se face pe loc
    - sunt string-uri care încep cu litera f sau F, iar valorile ce se doresc interpolate sunt folosite între acolade **{ }**.

```
print(F'For only {49:.2f} dollars! {"Have a great day!"}')
```

Liste

Week 2. Pythonic “Hello, World!”

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

Week 2. Pythonic “Hello, World!”

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

Week 2. Pythonic “Hello, World!”

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

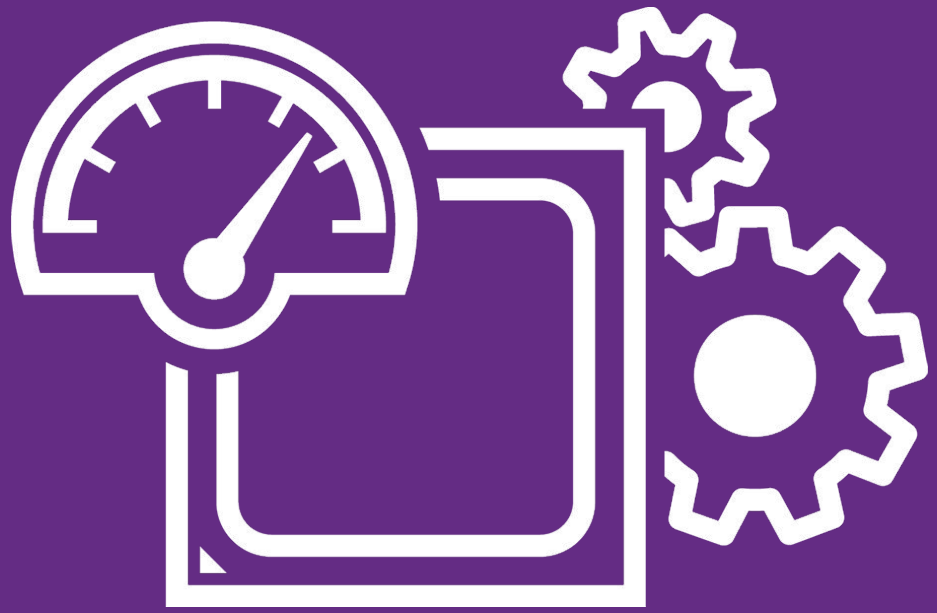
Week 2. Pythonic “Hello, World!”

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}
```

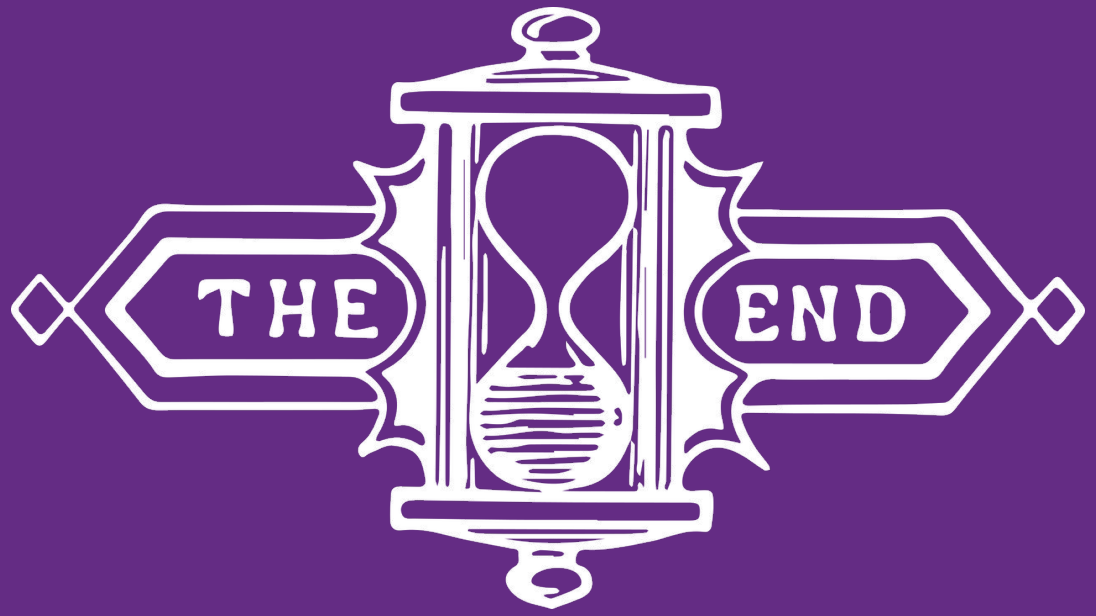
- Accesarea unui element din set nu este posibilă, dar setul poate fi parsat.
- Pentru eliminarea unui element din set se pot folosi metodele **remove()** sau **discard()**. Poate fi folosită și metoda **pop()**, dar această funcție va scoate din set ultimul element - setul fiind neordonat nu vom avea 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()**.



Temă

Temă

- La repository-ul creat săptămâna trecută adăugați un script Python care îndeplinește următoarele funcții:
    - declararea unei liste care să conțină elementele 7, 8, 9, 2, 3, 1, 4, 10, 5, 6 (în această ordine).
    - afișarea unei alte liste ordonată ascendent (lista inițială trebuie păstrată în aceeași formă)
    - afișarea unei liste ordonată descendent (lista inițială trebuie păstrată în aceeași formă)
    - afișarea numerelor pare din listă (folosind DOAR slice, altă metodă va fi considerată invalidă)
    - afișarea numerelor impare din listă (folosind DOAR slice, altă metodă va fi considerată invalidă)
    - afișarea elementelor multipli ai lui 3.
    - a se păstra acuratețea indexilor - aceștia trebuie să fie cât mai specifici.



Vă mulțumesc!