INFORMATICĂ · PYTHON · CLASA A IX-A · MEMORAREA DATELOR ȘI ORGANIZAREA CODULUI · LECȚIA 10

Introducere în programarea orientată pe obiecte: folosim clase predefinite

Durată: 50 min Nivel: mate-info · Python Înainte: lecțiile py9-02, py9-09

CE VEI ȘTI DUPĂ LECȚIE

- Explici, pe un exemplu, ce sunt clasa, obiectul, datele membre (atributele) și metodele.
- Creezi obiecte din clase predefinite (random.Random, datetime.date, fractions.Fraction) și le folosești membrii prin operatorul punct.
- Deosebești o metodă care întoarce un obiect nou de una care modifică obiectul (str față de list).
- Descoperi singur membrii unei clase cu type(), dir() și help().

1. Ai lucrat deja cu obiecte

Fiecare valoare din Python este un **obiect**, iar fiecare obiect aparține unei **clase** (tipul lui). Funcția `type()` îți spune clasa:

```
print(type("salut"), type(7), type(3.5), type([1, 2]))
# <class 'str'> <class 'int'> <class 'float'> <class 'list'>
```

Gândește-te la un model de telefon: **clasa** este descrierea modelului (ce date are un telefon și ce știe să facă), iar **obiectul** este telefonul tău concret, cu bateria lui și cu numărul lui.

Termen	Ce înseamnă	Exemplu
clasă	tipul, „planul” după care se construiesc obiectele	`str`, `list`, `datetime.date`
obiect (instanță)	o valoare concretă din acea clasă	`"salut"`, `[9, 7]`, data 8 septembrie 2026
dată membră (atribut)	o informație memorată în obiect	`d.year`, `f.numerator`
metodă	o funcție care aparține clasei și lucrează pe obiect	`s.upper()`, `note.append(8)`
bibliotecă (modul)	un fișier cu clase și funcții gata făcute, adus cu `import`	`random`, `datetime`, `fractions`

2. Operatorul punct: obiect.membru

Membrii unui obiect se accesează cu **punct**. La metode pui paranteze (cu argumente, dacă e cazul); la atribute, nu.

```
s = "informatica"
print(s.upper(), s.count("i"), s.find("ma"))   # INFORMATICA 2 5
print(s)                                       # informatica
```

Compară **funcția** `len(s)` (primește obiectul ca argument) cu **metoda** `s.upper()` (o „ceri" chiar obiectului). Ambele există; diferă doar felul de apel.

> **ATENȚIE — GREȘELI FRECVENTE**
>
> Dacă uiți parantezele, nu apelezi metoda, ci doar o „arăți": `print(s.upper)` afișează ceva de forma `<built-in method upper of str object at 0x…>`, nu textul cu majuscule.

3. Obiecte care se schimbă și obiecte care nu

Observă mai sus că `s` a rămas `informatica` după `s.upper()`. Obiectele `str` **nu se pot modifica**: metodele lor întorc un obiect **nou**. Obiectele `list` **se pot modifica**: multe metode lucrează direct pe listă și întorc `None`.

```
t = "ana"
u = t.upper()
print(t, u)                    # ana ANA

note = [9, 7, 10]
note.append(8)
print(note, len(note))         # [9, 7, 10, 8] 4
r = note.sort()
print(note, r)                 # [7, 8, 9, 10] None
```

Clasa	Se modifică obiectul?	Cum folosești rezultatul
`str`	nu	`t = t.upper()` (păstrezi obiectul nou)
`list`	da	`note.sort()` singur pe rând; **nu** `note = note.sort()`

> **SFAT**
>
> Clasa `list` o studiem pe larg în lecțiile py9-14 și py9-15 (o mini-introducere ai în lecția py9-09, secțiunea 3). Deocamdată reține doar ideea: înainte să folosești o metodă, află dacă **întoarce** ceva sau **modifică** obiectul.

4. Instanțierea: NumeClasă(argumente)

Ca să creezi un obiect dintr-o clasă, „chemi" clasa ca pe o funcție. Clasele din biblioteci le aduci întâi cu `import`.

random.Random este un generator de numere aleatoare. Dacă îi dai un număr de pornire (*seed*), produce de fiecare dată **același** șir de valori, util la teste:

```
import random
g = random.Random(2026)                  # instanțiere: obiectul g
print(g.randint(1, 6), g.randint(1, 6), g.randint(1, 6))   # 1 3 5
h = random.Random(2026)                  # alt obiect, același seed
print(h.randint(1, 6), h.randint(1, 6), h.randint(1, 6))   # 1 3 5
print(g.choice(["piatră", "foarfecă", "hârtie"]))          # hârtie
```

`g` și `h` sunt **două obiecte** din aceeași clasă; fiecare își ține propria stare (unde a ajuns în șir). Valorile de mai sus sunt cele obținute cu Python 3.13.

5. Datele membre: datetime.date

Un obiect `date` memorează trei atribute: `year`, `month`, `day`. Are și metode, iar operatorii `-` și `+` funcționează cu el.

```
import datetime
d = datetime.date(2026, 9, 8)            # an, lună, zi
print(d, d.year, d.month, d.day)         # 2026-09-08 2026 9 8
print(d.weekday())                       # 1 (0 = luni, deci marți)
print(d.strftime("%d.%m.%Y"))            # 08.09.2026
v = datetime.date(2027, 6, 18)
print((v - d).days)                      # 283
print(d + datetime.timedelta(days=100))  # 2026-12-17
```

Diferența a două date este un obiect din altă clasă, `timedelta`, care are atributul `days`. Atributele unei date **nu se pot schimba**: `d.year = 2030` dă `AttributeError`. Pentru altă dată creezi alt obiect.

6. Fracții exacte: fractions.Fraction

Numerele reale (`float`) sunt aproximative: `0.1 + 0.2 == 0.3` dă `False`. Clasa `Fraction` lucrează **exact**, cu numărător și numitor, și simplifică singură.

```
from fractions import Fraction
a = Fraction(1, 3)
b = Fraction(1, 6)
c = a + b
print(c, c.numerator, c.denominator)     # 1/2 1 2
print(Fraction(6, 8))                    # 3/4
print(Fraction(1, 10) + Fraction(2, 10) == Fraction(3, 10))   # True
print(float(Fraction(7, 8)))             # 0.875
```

Cu `from fractions import Fraction` scrii direct `Fraction(...)`; cu `import fractions` ai fi scris `fractions.Fraction(...)`.

Clasa	Instanțiere	Date membre	Metode / operații utile
`str`	`"text"` sau `str(123)`	(nu e cazul)	`upper()`, `count(x)`, `find(x)`, `split()`, `strip()`, `replace(a, b)`

Clasa	Instanțiere	Date membre	Metode / operații utile
`list`	`[1, 2]` sau `list(...)`	(nu e cazul)	`append(x)`, `count(x)`, `sort()`
`random.Random`	`random.Random(seed)`	(stare internă)	`randint(a, b)`, `choice(lista)`, `random()`
`datetime.date`	`datetime.date(an, lună, zi)`	`year`, `month`, `day`	`weekday()`, `strftime(...)`, `-`, `+ timedelta`
`fractions.Fraction`	`Fraction(p, q)`	`numerator`, `denominator`	`+ - * /`, `==`, `<`, `float(...)`

7. Cum descoperi singur alți membri

Nu trebuie să memorezi toate metodele. Le găsești:

- `type(x)`: din ce clasă este `x`;
- `dir(x)`: lista membrilor (ignoră-i pe cei care încep cu `_`);
- `help(datetime.date.weekday)`: descrierea unei metode, direct în interpretor;
- documentația oficială Python (docs.python.org), capitolul *Library Reference*.

DE ȘTIUT

Pentru curioși. Clasele proprii nu fac parte din programa clasei a IX-a. Doar ca să vezi de unde vin atributele și metodele, iată o clasă minimă:

```
class Punct:
    def __init__(self, x, y):       # se apelează la instanțiere
        self.x = x                   # date membre
        self.y = y
    def distanta_origine(self):      # metodă
        return (self.x ** 2 + self.y ** 2) ** 0.5

p = Punct(3, 4)
print(p.x, p.distanta_origine())     # 3 5.0
```

8. Verifică-te

1. În `d = datetime.date(2026, 12, 25)`, care este clasa, care este obiectul și ce dată membră are valoarea `12`?
2. Ce afișează `z = "mar"`, `z.upper()`, `print(z)`? De ce?
3. Ce valoare are `y` după `x = [3, 1, 2]` și `y = x.sort()`? Dar `x`?
4. Ce afișează `print(Fraction(2, 4) + Fraction(1, 4))`?
5. Ce zi a săptămânii este 25 decembrie 2026, știind că `datetime.date(2026, 12, 25).weekday()` dă `4`?

6. De ce sunt utile două obiecte `random.Random(5)` care dau aceleași valori?

Răspunsuri

1. Clasa este `datetime.date`, obiectul este `d`, iar `d.month` are valoarea `12`.
2. `mar`: `upper()` a creat un text nou, care nu a fost păstrat într-o variabilă; `str` nu se modifică.
3. `y` este `None`, iar `x` este `[1, 2, 3]`: `sort()` modifică lista și nu întoarce nimic.
4. `3/4`.
5. Vineri (0 = luni, 1 = marți, ..., 4 = vineri).
6. Poți repeta exact același „joc” sau test: rezultatul se poate verifica și compara, deși numerele par aleatoare.
