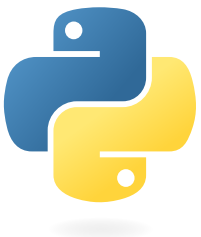
Llenguatges de Programació

Sessió 2: Python avançat



python™

Gerard Escudero i Albert Rubio

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Contingut

Classes I

Exemple de classe:

```
class Treballador:
    treCompt = 0                # atribut de classe:
                                # un únic valor per classe
    def __init__(self, nom, salari):  # constructor
        self.nom = nom          # definició d'atribut
        self.salari = salari    # self representa el objecte creat
        Treballador.treCompt += 1

    def getCompt(self):         # definició de mètode
        return Treballador.treCompt

    def getSalari(self):
        return self.salari
```

Exemple d'interacció:

```
tre1 = Treballador("Pep", 2000)
tre2 = Treballador("Joan", 2500)
tre1.getSalari()  👉  2000
tre1.salari  👉  2000
tre1.getCompt()  👉  2
Treballador.treCompt  👉  2
```

Classes II

Podem afegir, eliminar o modificar atributs de classes i objectes en qualsevol moment:

```
tre1.edat = 8
tre1.edat  👉  8
hasattr(tre1, 'edat')  👉  True
del tre1.edat
hasattr(tre1, 'edat')  👉  False
```

La comanda `dir` ens retorna la llista d'atributs d'un objecte. Hi ha molts predefinits:

- `__dict__`, `__doc__`, `__name__`, `__module__`, `__bases__`.

Atributs ocults

```
class Treballador:
    __treCompt = 0          # atribut ocult: començant per 2 _

tre1.__treCompt  ❌
tre1.tre1._Treballador__treCompt  👉  2
```

Herència

Exemple:

```
class Fill(Treballador):    # pare entre ( )
    def fillMetode(self):
        print('Cridem al metode del fill')

tre3 = Fill('Manel', 1000)
tre3.fillMetode()
```

Podem tenir herència múltiple:

```
class A: # definim la classe A
.....
class B: # definim la calsse B
.....
class C(A, B): # subclasse de A i B
.....
```

Per fer comprovacions:

- `issubclass(sub, sup)`
- `isinstance(obj, Class)`

Contingut

- Classes
- Tipus algebraics
- *Lazyness*
- Exercicis

Tipus enumerats

Donen la llista de valors possibles dels objectes:

Pedra, paper, tisores:

```
class Pedra:
    pass

class Paper:
    pass

class Tisores:
    pass

Jugada = Pedra | Paper | Tisores

guanya = lambda a, b: (a, b) in [(Paper,Pedra),(Pedra,Tisores),(Tisores,Paper)]
```

Exemple:

Guanya la primera a la segona?

```
guanya(Paper, Pedra)  👉  True
```

Tipus algebraics

Defineixen constructors amb zero o més dades associades:

```
from dataclasses import dataclass

@dataclass
class Rectangle:
    amplada: float
    alçada: float

@dataclass
class Quadrat:
    mida: float

@dataclass
class Cercle:
    radi: float

class Punt:
    pass

Forma = Rectangle | Quadrat | Cercle | Punt
```

- El decorador `dataclass` defineix automàticament mètodes com `__init__`.

Funcions amb tipus algebraics

Declaració:

```
from math import pi

def area(f):
    match f:
        case Rectangle(amplada, alçada):
            return amplada * alçada
        case Quadrat(mida):
            return area(Rectangle(mida, mida))
        case Cercle(radi):
            return pi * radi**2
        case Punt():
            return 0
```

- `match` permet reconèixer patrons

Crida:

```
area(Quadrat(2.0))  👉  4.0
```

Tipus recursius

Arbre binari:

```
from __future__ import annotations
from dataclasses import dataclass

class Buit:
    pass

@dataclass
class Node:
    val: int
    esq: Arbre
    dre: Arbre

Arbre = Node | Buit
```

```
def mida(a: Arbre) -> int:
    match a:
        case Buit():
            return 0
        case Node(x, e, d):
            return 1 + mida(e) + mida(d)
```

```
t = Node(1,Node(2,Buit(),Buit()),
           Node(3,Buit(),Buit()))

mida(t)  👉  3
```

- `annotations` permet els tipus recursius (en aquest cas `Arbre`).

Contingut

- Classes
- Tipus algebraics
- *Lazyness*
- Exercicis

Iteradors

En python existeixen mols tipus que són iterables:

- strings, llistes, diccionaris i conjunts
- definits per l'usuari.

El protocol d'iteració el defineixen els mètodes:

- `__iter__` i `next`

i llença l'excepció `StopIteration` en acabar:

```
s = 'abc'
for c in s:
    print(c)
```

👉

```
a
b
c
```

```
s = 'abc'
it = iter(s)
while True:
    try:
        print(it.__next__())
    except StopIteration:
        break
```

Generadors I

Són el mecanisme que permeten l'avaluació *lazy* en python3.

Exemple: sèrie de fibonacci fins a un nombre determinat.

```
def fib(n):
    a = 0
    yield a
    b = 1
    while True:
        if b <= n:
            yield b
            a, b = b, a + b
        else:
            raise StopIteration
```

```
f = fib(1)
next(f)  👉  0
next(f)  👉  1
next(f)  👉  1
next(f)  ❌  StopIteration

## La comanda yield para l'execució
## de la funció fins a la següent
## invocació de next.


## amb list comprehension
[x for x in fib(25)]
👉
[0, 1, 1, 2, 3, 5, 8, 13, 21]
```

Generadors II

Els generadors poden ser infinits:

```
def fib2():
    a = 0
    yield a
    b = 1
    while True:
        yield b
        a, b = b, a + b
```

```
f = fib2()
[next(f) for _ in range(8)]
```

👉

```
[0, 1, 1, 2, 3, 5, 8, 13]
```

Generadors III

classe amb generador:

```
class fib:
    def __iter__(self):
        a, b = 0, 1
        while True:
            yield b
            a, b = b, a + b
```

```
for (i, x) in enumerate(fib(), 1):
    if i > 10:
        break
    print(i, x)
```

👉

```
1 1
2 1
3 2
4 3
5 5
6 8
```

classe amb generador i iterador:

```
class fib2:
    def __init__(self):
        self.gen = self.__iter__()
```

```
    def __next__(self):
        return next(self.gen)
```

```
    def __iter__(self):
        a, b = 0, 1
        while True:
            yield b
            a, b = b, a + b
```

```
f = fib2()
print([next(f) for _ in range(6)])
```

👉 `[1, 1, 2, 3, 5, 8]`

Són útils quan volem tenir varies instàncies del generador funcionant a l'hora.

Contingut

- Classes
- Tipus algebraics
- *Lazyness*
- Exercicis

Exercicis

- Feu aquests problemes de Jutge.org:
  - P71608 Classe per arbres
  - P45231 Generadors
  - P53498 Definició d'un iterable