Wstęp do Informatyki

Dla Bioinformatyków

Wykład 9:
Wyjątki i Obiekty

Bartek Wilczyński

14.12.2015

Plan na dziś

- Słów kilka o historii jezyków obiektowych
- Wszystko w pythonie jest obiektem
- Klasy a typy (Jak rozpoznać typa wśród klasy?)
- O problemach dziedziczenia
- Metody specjalne i operatory
- O sprawach prywatnych i publicznych
- Sytuacje wyjątkowe nie są wcale takie wyjątkowe

Kilka nazw z historii

- Simula 67 - Ole-Johan Dahl, Kristen Nygaard, rozszerzenie języka Algol 60
- Bezpośredni wpływ na języki strukturalno-obiektowe tj. C++ (1979,1983), Java (1995), itp.
- Równolegle rozwój dynamicznych języków obiektowych: Smalltalk (1972,1980), Eiffel (1985)
- Obecnie także bardzo popularne są języki skryptowe czerpiące z obu wzorców: python (1991) i ruby (1995)

Obiekty w pythonie

- Mówiliśmy o różnych typach wartości w pythonie: liczby, napisy, słowniki, listy, wyrażenia regularne, etc.
- pamiętamy, że zmienne nie mają typu, ale wartości tak
- Okazuje się, że typy zmiennych też mają swój typ – są klasami
- Klasy z kolei są obiektami typu `type` (kto jeszcze się nie zgubił?)

Jak sprawdzić typa?

- Możemy używać funkcji wbudowanej `type`
  np: `type(5)->int, type(“ala”)->str, type([1])->list`
- Co ciekawe, typy zmiennych mają też swój typ:
  `type(int)->type, type(str)->type, type(type)->type`
- Zwróćmy uwagę zwłaszcza na tę ostatnią linijkę: type(type)==type. Czy to nie za dużo typów?
- No i jak to działa, że wywołujemy te typy zupełnie jak funkcje?

Skąd obiekty wiedzą jakiego są typu?

- Przyjrzyjmy się atrybutowi __class__:

```
In [23]: "aaa".__class__
Out[23]: str

In [24]: str.__class__
Out[24]: type

In [25]: type(str)
Out[25]: type

In [26]: type("aaa")
Out[26]: str
```

- Przy okazji zauważmy nowe słowo **atrybut**

Obiekty, atrybuty, metody

- Każdy obiekt ma swoją przestrzeń nazw

```
In [28]: f=open("/etc/passwd")

In [29]: f.
f.close        f.flush        f.next         f.seek         f.writelines
f.closed       f.isatty       f.read         f.softspace    f.xreadlines
f.encoding     f.mode         f.readinto     f.tell
f.errors       f.name         f.readline     f.truncate
f.fileno       f.newlines     f.readlines    f.write
```

- Jeśli atrybut jest funkcją nazywamy go **metodą**

```
In [36]: f.mode
Out[36]: 'r'

In [37]: f.readline
Out[37]: <function readline>

In [38]: f.readline()
Out[38]: ''
```

Co prywatne, co publiczne

- Atrybuty rozpoczynające się od __ uważane są za prywatne, ale w pythonie jest to tylko konwencja

```
In [27]: l=[1,2,3]

In [28]: l.
l.append   l.extend   l.insert   l.remove   l.sort
l.count    l.index    l.pop      l.reverse

In [28]: l._
l.__add__           l.__getslice__      l.__new__
l.__class__         l.__gt__            l.__reduce__
l.__contains__      l.__hash__          l.__reduce_ex__
l.__delattr__       l.__iadd__          l.__repr__
l.__delitem__       l.__imul__          l.__reversed__
l.__delslice__      l.__init__          l.__rmul__
l.__doc__           l.__iter__          l.__setattr__
l.__eq__            l.__le__            l.__setitem__
l.__format__        l.__len__           l.__setslice__
l.__ge__            l.__lt__            l.__sizeof__
l.__getattribute__  l.__mul__           l.__str__
l.__getitem__       l.__ne__            l.__subclasshook__
```

Czy możemy stworzyć nową klasę?

```
class Kwardat:
    bok=1
    def pole(self):
        return self.bok**2
```

- Możemy teraz użyć klasy Kwadrat, żeby stworzyć **instancję** k:

```
In [49]: k=Kwadrat()

In [50]: k
Out[50]: <__main__.Kwadrat instance at 0x7f14d1e495a8>

In [51]: Kwadrat.bok
Out[51]: 1

In [52]: k.bok
Out[52]: 1

In [53]: k.pole
Out[53]: <bound method Kwadrat.pole of <__main__.Kwadrat instance at 0x7f14d1e495a8>>

In [54]: Kwadrat.pole
Out[54]: <unbound method Kwadrat.pole>
```

Co się dzieje z atrybutami instancji?

- Zaczynamy z atrybutem bok=1 w klasie Kwadrat

```
In [63]: k.pole()
Out[63]: 1

In [64]: k.bok=5

In [65]: k.pole()
Out[65]: 25

In [66]: Kwadrat.bok
Out[66]: 1

In [67]: k2=Kwadrat()

In [68]: k2.pole()
Out[68]: 1
```

Coś o dziedziczeniu

- Klasy mogą dziedziczyć po sobie atrybuty I metody

```
import math
class Foremny():
    bok=1
    def pole(self):
        r=self.bok/(2*math.tan(math.pi/self.n))
        return r*self.bok*self.n/2

class Kwadrat(Foremny):
    n=4

class Rownoboczny(Foremny):
    n=3
```

Używanie instancji podklas

- Używając podklas, możemy liczyć pola różnych figur

```
In [94]: k=Kwadrat()

In [95]: k.bok=4

In [96]: k.pole()
Out[96]: 16.000000000000004

In [97]: t=Rownoboczny()

In [98]: t.bok=4

In [99]: t.pole()
Out[99]: 6.928203230275511
```

Specjalna metoda __init__

- Metoda __init__(self, ….) pozwala na inicjowanie instancji:

```
class Foremny_konst(Foremny):
    def __init__(self, bok):
        self.bok=bok
```

- Możemy to wykorzystać do inicjalizacji wartości:

```
class Kwadrat_konst(Foremny_konst):
    n=4
```

- I potem tego użyć:

```
In [137]: kk=Kwadrat_konst(4)

In [138]: kk.pole()
Out[138]: 16.000000000000004
```

Dziedziczenie z wielu stron

- Możemy dziedziczyć z wielu klas
- Dzięki temu dziedziczymy atrybuty i metody ze wszystkich nadklas
- Kolejność dziedziczenia ma znaczenie

```
In [139]: class Trojkat_konst(Foremny_konst,Rownoboczny):
   .....:     pass
   .....:

In [140]: tt=Trojkat_konst(bok=5)

In [141]: tt.pole()
Out[141]: 10.825317547305485
```

Sytuacje wyjątkowe

- Możemy reagować na sytuacje “wyjątkowe” poprzez przechwytywanie wyjątków

```
try:
    f=open("/etc/passwd","w")
except IOError:
    print "nie mogę otworzyć pliku"
else:
    print "Sukces - skasowałem plik /etc/passwd"
```

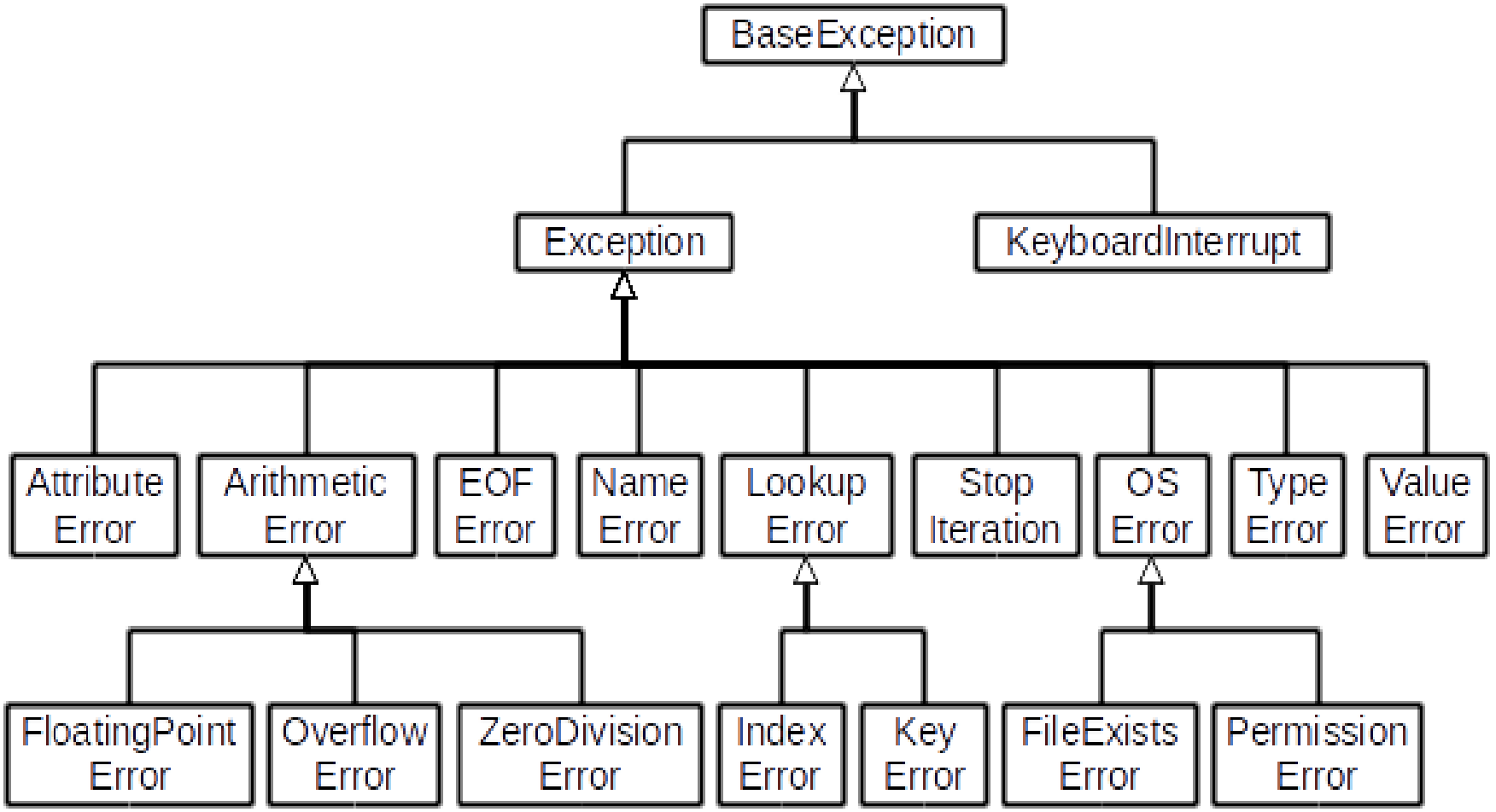
- Możemy też zgłaszać wyjątki

```
if self.bok==0:
    raise ValueError
```

Podsumowanie - obiekty

- Klasy są uogólnieniem typu obiektów, a instancje są w pewnym sensie uogólnieniem wartości
- Atrybuty i metody(atrybuty funkcyjne) są definiowane na poziomie klasy a używane zwykle na poziomie instancji
- Prywatne atrybuty mają nazwy rozpoczynające się od __ i niepowinny być wywoływane spoza danej definicji klasy (w pythonie jest to konwencja)
- Dziedziczenie pomiędzy klasami pozwala na używanie w podklasach atrybutów i metod zdefiniowanych w nadklasie

Hierarchia wyjątków w pythonie



Podsumowanie - wyjątki

- Możemy zgłaszać wyjątki różnych rodzajów przy pomocy raise
- Te wyjątki mogą być “chwytane” przy pomocy klauzuli try: … except: …. w sposób specyficzny (z podaniem typu wyjątku lub bez)
- Ten proces działa kaskadowo, jeśli jedna klauzula try nie złapała tego konkretnego wyjątku jest on przekazywany “wyżej”
- Wyjątki “nie złapane” powodują zakończenie programu
- Można konstruować własne klasy dziedziczące z istniejących klas wyjątków
- Można chwytać nadklasy wyjątków
  np. except ArtithmeticError: