# Objekty a třídy (nejen v Pythonu) znovu a podrobněji

# Objekty a třídy (nejen v Pythonu)

## Pojmy:

Třída je datový typ (formička)

Objekt je instance (bábovička)

TH je objekt třídy Člověk

# Jak je to v jazyku Python

## Definice třídy

```
class Komplex:
    pass  # dál už nic není

k = Komplex()
k.Re = 3.00
k.Im = 4.00
print( k.Re, k.Im )

3.0 4.0
```

# Přidat funkci

- má povinný první parametr `self` = TENHLE objekt, který (ten parametr) se při volání neuvádí

```
class Komplex:
    def abs( self ):
        return ( self.Re*self.Re
                 +self.Im*self.Im )**(1/2)
k = Komplex()
k.Re = 3.00
k.Im = 4.00
print( k.Re, k.Im, k.abs() ) # bez parametru!
```

```
3.0 4.0 5.0
```

# Konstruktor

- funkce pro inicializaci nového objektu
- volá se při vytváření objektu
- vždycky se jmenuje `__init__`
- …takže může být jenom jeden
- má také povinný první parametr `self`
- když konstruktor neuvedeme,
  použije se standardní (prázdný) konstruktor

# Konstruktor

```
import math

class Komplex:
    def abs( self ):
        return ( self.Re*self.Re
                 +self.Im*self.Im )**(1/2)
    def __init__( self, Re, Im ):
        self.Re = Re
        self.Im = Im


k = Komplex( 3.00, 4.00 )
print( k.Re, k.Im, k.abs() )
```

```
3.0 4.0 5.0
```

# Dědičnost

- Při definici nové třídy můžeme říci,
  že je odvozená (zděděná) od jiné třídy.
- Odvozená třída zdědí:
  - všechny datové součásti
  - všechny funkce
- Navíc můžeme definovat nová data i nové funkce
- Zděděné funkce můžeme změnit (předefinovat)

# Dědičnost

```
class A:
    x = 10
    def tiskA(self):
        print( "tiskA()" )

class B( A ):
    def tiskB(self):
        print( "tiskB() x=", self.x ) #
        self.tiskA()

b = B()
b.tiskB()
```

```
tiskB() x= 10
tiskA()
```

# Dědičnost - volání funkce předka

```
class A:
    def f(self):
        print( "A.f()" )

class B(A):
    def f(self):
        print( "B.f()" )
        super().f()

class C(B):
    def f(self):
        print( "C.f()" )
        super().f()

c = C()
c.f()
```

```
C.f()
B.f()
A.f()
```

# Může objekt patřit do více tříd?

Hierarchické vztahy mezi třídami ("is a"):

pes je savec
savec je obratlovec (takže pes je taky obratlovec)
obratlovec je živočich (takže pes je taky živočich)

Ale nejenom to:

pes je savec
pes je domácí zvíře (a ne každé domácí zvíře je savec)
pes je přítel člověka (nejlepší)

TH je Učitel, Programátor, Muzikant, Vynálezce, Výtvarník…

# Násobná dědičnost

v různých programovacích jazykách
lze deklarovat, že třída je pod-třídou jiné třídy
(„dědičnost"), potom dědí všechny prvky svého předka

v některých programovacích jazykách (i v Pythonu)
lze deklarovat, že třída je pod-třídou více tříd
(„násobná dědičnost"), potom dědí všechny prvky všech
svých předků.

Je s tím spousta potíží !! („diamond problem")

proto se v některých programovacích jazykách
(násobná) dědičnost nahrazuje pomocí interface

# Interface („rozhraní")

Na rozdíl od dědičnosti se nic nedědí,
jenom slibujeme, že objekt tyhle funkce bude mít/umět
(„objekt splňuje interface XYZ")

Abstraktní datový typ

V Pythonu – viz Duck typing (dále).

# Polymorfismus

```
class Zvire:
    def __init__(self, jmeno):
        self.jmeno = jmeno
    def VydejZvuk(self):
        print("!@#$%^")

class Pes(Zvire):
    def VydejZvuk(self):
        print("HAF:", self.jmeno)

class Kocka(Zvire):
    def VydejZvuk(self):
        print("Mnau:", self.jmeno)

class Had(Zvire):
    def VydejZvuk(self):
        print("Sssss:", self.jmeno)

class Kapr(Zvire):
    def VydejZvuk(self):
        print("...:", self.jmeno)

zoo = [ Pes("Archie"), Kocka("Babeta"), Had("Python"), Kapr("Karel") ]
for z in zoo:
    z.VydejZvuk()
```

```
HAF: Archie
Mnau: Babeta
Sssss: Python
...: Karel
```

# Abstraktní třída

- Třída, ze které nechceme vytvářet instance
- Slouží jenom jako společný předek
- Python sám neumí, řeší modul abc (Abstract Base Classes)
- V jazycích s typovou kontrolou potřebná ke kontrole, jestli předáváme parametr správného/odpovídajícího typu
- V Pythonu se nekontroluje !@#$%^ (dynamické typování)…

# Duck typing:

Když to chodí jako kachna, plave jako kachna
a kváká jako kachna… – tak je to kachna!

# Kontrola typu

1. funkce type(…)

```
if type(th) != Herec: # Herec je jméno třídy
    print("vetřelec!!!")
```

2. funkce isinstance(…)

```
if not isinstance(p, Zvire):
    print("p není Zvíře!")
```

isinstance(…) vrací hodnotu True i pro odvozené typy

…ale v Pythonu se používá duck-typing.

*) typové anotace – někdy jindy

# Zapouzdření (encapsulation)

Možnost určovat,
které součásti objektu budou viditelné zvenku (veřejné)
a které budou viditelné jenom uvnitř (soukromé).

Slouží pro zachování konsistence dat.

„Nedovolte nikomu sahat na svá data!"

…v Pythonu není.

- Náhražka:Když jméno začíná _ (podtržítko),
  tak byste to (u cizího objektu) neměli používat/volat
  (a neimportuje se příkazem import)
- Když jméno začíná __ (dvě podtržítka),
  a na konci má max. jedno podtržítko
  tak se __jméno přejmenuje na _JménoTřídy__jméno
  (tím se zabrání konfliktům jmen s rodičovskou třídou)

```
class Trida:
    __TajnaPromenna = 27

t = Trida()
print( t._Trida__TajnaPromenna )
```

# Komu patří funkce...

## Funkce definovaná v třídě může být

1) funkce instance/objektu
2) statická funkce
3) funkce třídy

## 1) Funkce patřící objektu

- patří objektu (volá se prostřednictvím objektu)
- má přístup k jeho funkcím a proměnným

= to, co jsme viděli doposud

# 2) Statická metoda/funkce

- patří třídě a ne objektu
  (volá se pomocí třídy i objektu)
- nemá přístup k proměnným objektu
  (žádný objekt ani nemusí existovat)

```
class Trida:

    @staticmethod
    def sedm():
        return 7

print( Trida.sedm() )          7
t = Trida()
print( t.sedm() )              7
```

# 3) Metoda třídy (class method)

- patří třídě a ne objektu (volá se prostřednictvím třídy)
- nemá přístup k proměnným objektu
  (žádný object ani nemusí existovat)
- má přístup k proměnným třídy

```
class AA:
    pocet = 0
    @classmethod
    def MetodaTridy(cls): # parametr cls
        cls.pocet += 1
        print( cls.pocet )
```

```
AA.MetodaTridy()        1
AA.MetodaTridy()        2
```

# Komu patří proměnné...

## Proměnná definovaná v třídě může být

1) proměnná patřící instanci
2) proměnná patřící třídě

# 1) proměnná patřící instanci

Zatím všechno, co jsme viděli,
i když ve skutečnosti…

# 2) proměnná patřící třídě

```
class KLAS:
    pocet = 0
    def __init__(self):
        KLAS.pocet += 1
    def f(self):
        #self.pocet += 1
        return self.pocet

t = KLAS()
print( t.f(), t.f(), t.f() )
s = KLAS()
print( s.f(), s.f(), s.f() )
```

a)
```
#self.pocet += 1:

1 1 1
2 2 2
```

b)
```
self.pocet += 1:

2 3 4
3 4 5
```

Pokud v tom nemáte jasno, tak to nepoužívejte!

# Virtuální metody

V odvozené třídě lze předefinovat metodu tak,
že všechna její volání (i ze starších tříd a metod)
budou volat tu novou.

V Pythonu jsou všechny metody virtuální.

Podrobnosti – u nějakého jiného jazyka.

# Někdy jindy, případně vůbec

objektový návrh…

dekorátory

class-factory

# Funkce dir()

dir( )

seznam jmen v aktuálním prostoru
(globální nebo uvnitř funkce lokální)

dir( objekt )

seznam atributů obsažených a dosažitelných
v daném objektu

```
>>> dir()
['__annotations__', '__builtins__', '__doc__',
'__loader__', '__name__', '__package__',
'__spec__']
```