Složitější proměnné

- OBJEKT zelva.forward(100)
- STRING (textový řetězec) "abcd"
  neměnitelná
- TUPLE (n-tice) ("NPRG031", 1)
  neměnitelná
- LIST (seznam) [1, 2, 5, 3, 2]
  seznam s pořadím
- SET (množina) { 1, 5, 2 }
  bez pořadí
  každá hodnota je nebo není (max jednou)
- DICTIONARY (slovník) { 1:"a", 5:"xyz", 2:22 }
  dvojice klíč:hodnota

Dynamické proměnné

- Proměnné, které vznikají (a zanikají) během výpočtu
- Mohou mít různou velikost
- Zvláštní oblast paměti – halda
- Nepotřebné proměnné se automaticky uklízejí
  (ne ve všech jazykách, ale v Pythonu ANO)
  = garbage collector
- Proměnná, která vypadá, že obsahuje třeba seznam, ve skutečnosti obsahuje pouze odkaz (adresu do haldy)
- Příklad: list, string, (dlouhé) celé číslo, slovník, objekt
- …ale také spojový seznam (vytvořený z objektů) nebo strom (taky vytvořený z objektů)

Objekt (podrobnější pohled)

Obsahuje DATA a FUNKCE

Přístup k nim pomocí tečky

```
>>> a = 5
>>> a.bit_length()
3
>>> zelva.left(90)
>>> zelva.DEFAULT_ANGLEOFFSET
0
```

Skoro všechno v Pythonu je objekt.

Proč objekty (nejen v Pythonu)

- Rozdělení programu na části, organizace kódu
- Další CO to dělá x JAK to dělá
- Řeší problémy, na které jsme zatím potřebovali globální proměnné (číst vstup, seznam slov…)
- Lze chápat i tak, že obsahují dodatečné parametry svých funkcí…

Objekt obsahuje DATA a FUNKCE

…a ne vše, co obsahuje, je veřejné!

Objekty a třídy (nejen v Pythonu)

Pojmy:

Třída je datový typ (formička)

Objekt je instance (bábovička)

TH je objekt třídy Člověk

Přirozená otázka:

Může být nějaký objekt instancí více tříd?

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 …analogicky)

Ale nejenom to:

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

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

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).

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

```
import math

class Komplex:
    def abs( self ):
        return math.sqrt( self.Re*self.Re
                          + self.Im*self.Im )
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 math.sqrt( self.Re*self.Re
                          + self.Im*self.Im )
    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
```

Příklad - seznam slov (pomocí listu)

```
class Seznam:
    seznam = []
    def Pridej( self, slovo ):
        for d in self.seznam:
            if d[0]==slovo:
                d[1] += 1
                return
        self.seznam.append( [ slovo,1 ] )
```

```
s = Seznam()
for slovo in "a b a b c a".split():
    s.Pridej( slovo )
print( s.seznam )
```

```
[['a', 3], ['b', 2], ['c', 1]]
```

Příklad - seznam slov (pomocí slovníku)

```
class Seznam2:
    seznam = {}
    def Pridej( self, slovo ):
        if not slovo in self.seznam:
            self.seznam[slovo] = 0
        self.seznam[slovo] +=1
```

```
s = Seznam2()
for slovo in "a b a b c a".split():
    s.Pridej( slovo )
print( s.seznam )
```

```
{'a': 3, 'b': 2, 'c': 1}
```

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()
```

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 isinstance(p, Zvire):
    print("p není Zvíře!")
```

isinstance(…) vrací hodnotu True i pro odvozené typy

…ale v Pythonu se používá duck-typing.

Viditelné a neviditelné atributy a funkce

Potřeba k zapouzdření (encapsulation)

…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),
  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 )
```

```
27
```