Python - skripty

Skript je skupina po sobě jdoucích příkazů uložená do souboru. Původně krátký program pro příkazový interpret OS

Z důvodu strukturování je lépe vytvořit v souboru řídící fci a tu v tom souboru hned vyvolat

Skripty se obvykle spouští z příkazového řádku (častější v Unixu než ve Windows)

```
#Pr. skript1.py ,
# spust ve Windows: c:\sw\python25>python d:pgs\python\skript1.py,
# nebo kratsi zapis, daš-li skript do direktory  s python.exe, staci …>python skript1.py
# definice fce main.            Může mít i jiné jméno.
def main():
    print 'bezi skript 1'
    jmeno = raw_input('jak se jmenujes? : ')
    print 'Ahoj ', jmeno

# vyvolani fce main
main()
raw_input('skonci stiskem ENTER')
                                        # Ctrl C zastavi beh skriptu
```

Python - skripty

Z příkazového řádku lze předávat argumenty. Ty jsou ulozeny jako seznam v systémové proměnně sys.argv

```
# Pr. skript2.py
import sys
def main():
    print "pri spusteni pridane argumenty, ulozi se do sys.argv"
    print sys.argv

main()
```

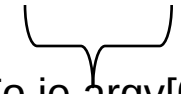
Spustíme zápisem v cmd okně, **analogický způsob s argumenty již v IDLE nelze provest**

C:\sw\python25>python skript2.py ar1 ar2 ar3

Vypise se :

pri spusteni pridane argumenty, ulozi se do sys.argv

[‘skript2.py’, ‘ar1’, ‘ar2’, ‘ar3’]



To je argv[0]

Python - skripty

Skript může akceptovat z příkazového řádku i přepínače (volby), stejně jako argumenty.
K tomu je vhodné použít modul *getopt* , který obsahuje podporu pro synt. analýzu řetězce přepínačů, které chce skript rozpoznávat a argumentů. Funkce getopt vrací dva seznamy. První tvoří n-tice nalezených přepínačů ve tvaru (přepínač, jeho parametr) druhý vrací normální argumenty.

```
# skript3.py
import getopt, sys
                                        přiřadí od argv[1];    mustr pro 4 optios
def main():
   (volby, argumenty) = getopt.getopt(sys.argv[1:], 'f:vx:y:')    # je-li dvojtecka za znakem,
                                                                   #tak prepinac vyzaduje parametr
   print 'volby:', volby
   print 'argumenty:', argumenty

main()
```

dvojice přepínačů a jejich parametrů
pojmenování dovolí volné pořadí při spouštění

Spuštění

…python skript3.py –x100 –v –y50 –f nejakySoub ar1 ar2
argumenty

Python - skripty

Podporu pro zpracování řádků ze vstupních souborů dává modul *fileinput .* Z proměnné sys.argv načte argumenty příkazového řádku a použije je jako jména vstupních souborů, které po řádcích zpracuje

```
# skript4.py            vypouští řádky začínající komentářem a tiskne počet radek
import fileinput
def main():                          #čte řádky souborů daných argv[1], argv[2], …
    for radek in fileinput.input():
        if fileinput.isfirstline():      #pozna zacatek souboru
            print 'soubor %s' % fileinput.filename()
        if radek[:1] != '#':             #znak s indexem [0]
            print radek
    print 'pocet radek ', fileinput.lineno()     #lineno() je pocet radek za vsechny soubory
                          # další fce viz modul fileinput
main()
```

Spuštění např.

```
C:\sw\python25>python skript4.py skript3.py skript2.py
                              argument1                    argument2
```

Python - skripty

Spouštět skripty ve Windows lze i dalšími způsoby:

-Přesuneme se v MSDOS okně do adresáře se skripty (napr. D:\pgs\python a v příkazovém řádku napíšeme např.

c:\sw\python25>python.exe skript1.py

-Na soubor se skriptem klepneme prav tl., vybereme Odeslat na plochu (vytvořit zástupce).
Na vzniklou ikonu klepneme prav. tl., vybereme vlastnosti a v nich lze nastavit adresář,
doplnit argumenty a zavést klávesovou zkratku stiskem ctrl + písmeno. Tou lze pak skript spouštět
Pokud má skript parametry, lze je uvést v poli Cíl
Lze spustit i poklepáním na ikonu.

-Spustit lze také Start – Spustit a do dialogového okna zapsat např.
python skript1.py
program se ale skončí zavřením okna (zůstane otevřené při uvedení přepínače -i)

Python - skripty

Přesměrování vstupů / výstupů lze provést z příkazového řádku

Např.

c:\sw\python25>python skript2.py <skript1.py >out.txt

Skript2.py bude číst ze souboru skript1.py a zapisovat do souboru out.txt

Čtení a zápis lze provádět rovněž s použitím modulu **sys**

Př. skript5.py

```
# skript5.py
import string, sys
def main():
    sys.stdout.write(string.replace(sys.stdin.read(), # řetězec daný 1.argumentem bude nahrazen
                                sys.argv[1], sys.argv[2] ))   # řetězcem daným druhým argumentem
main()
```

Normálně spuštěn by pracoval s klávesnicí a obrazovkou. Spustíme ho s přesměrováním

c:\sw\python25>python skript5.py main hlavni <skript1.py >nic.txt

Python - skripty

Krátkým skriptům postačuje jediná funkce.

U rozsáhlých skriptů je vhodné oddělit řídící funkci main od dalších funkcí

Př skript6.py

Provádí překlad dvouciferných čísel do slovního tvaru.

Řídící fce main( ) volá fci **prekladDo99** se zadaným argumentem (viz %)

Skript voláme z příkazového řádku např.

… **python skript6.py 23**

Python - skripty

```
import sys
do9 = {'0':'', '1':'one', '2':'two', '3':'three', '4':'four' }                #atd.
od10do19 = {'0':'ten', '1':'eleven', '2':'twelve', '3':'thirteen'}            #atd
od20do90= {'2':'twenty', '3':'thirty', '4':'fourty', '5':'fifty'}             #atd

def prekladDo99(cifernyTvar):
    if cifernyTvar == '0': return('zero')
    if len(cifernyTvar) > 2:
        return "vice nez dvouciferne cislo"
    cifernyTvar = '0' + cifernyTvar
    decades, units = cifernyTvar[-2], cifernyTvar[-1]
    if decades == '0': return do9[units]
    elif decades =='1': return od10do19[units]
    else: return od20do90[decades]+' '+do9[units]

def main():
    print prekladDo99(sys.argv[1])

main()
```

Python - skripty

Skripty lze použít jako moduly, chceme-li jejich kód spustit v jiném skriptu nebo modulu.

Tuto možnost zajistí podmíněné volání řídící fce main( )

```
if  __name__  == '__main__' :       #__name__ je atributem obsahujicim jmeno fce
    main( )
else
    #  případný inicializační kód modulu
```

Bude-li soubor volán jako skript, bude mít proměnná __name__ hodnotu __main__,

Je-li soubor importován jako modul do jiného modulu, obsahuje proměnná

__name__ název souboru

Použití ukazuje př.skript7.py a skriptJakoModul.py (viz%)

Také naopak, modul může být upraven tak, aby mohl být spuštěn jako skript. K tomu stačí když v proměnné __name__ zajistí, že obsahuje hodnotu '__main__'

Python - skripty

Př.skript7.py

. . .

```
# az sem to je stejne se skript6.py
def main():
    print prekladDo99(sys.argv[1])

if __name__ == '__main__':
    main()
else: print __name__, 'je zaveden jako modul'
```

Vyvolat c:\sw\python25>python skript7.py 12

Př skriptJakoModul.py

```
import skript7
#delej si co chces
c = '12'
print skript7.prekladDo99(c)
```

Python třídy a objekty

class C(R0, R1, …) : # dovoluje vice rodicu

B_i # blok proveden jednou při zpracování definice.

pomocí přiřazování vytvoří proměnné třídy a zpřístupni je *třída.proměnná*

def m_0 (self, …) : #metoda

B_0 #blok

def m_1 (self, …) :

B_1

…

Jméno konstruktoru je vždy __init__(parametry), v potomkovi se musí explicitně volat
V konstruktoru potomka je nutné explicitně volat konstruktor jeho rodiče příkazem

rodič.__init__(self, případné další parametry)

Proměnné instance jsou vytvářeny přiřazovacím příkazem uvnitř konstruktoru
Privátní metody a proměnné instancí tříd jsou pojmenovány **__jméno**
a jsou použitelné jen uvnitř třídy.

Probíhá-li výpočet uvnitř metody třídy, má přístup do

lokálního prostoru jmen = argumenty a proměnné deklarované v metodě
globálního prostoru jmen = funkce a proměnné deklarované na úrovni modulu
vestavěného prostoru jmen = vestavěné funkce a výjimky

Python třídy a objekty

Př.P4Obrazce (Spustit bud v Idle nebo v d:\pgs\python>c:\sw\Python25\python a pak >>>import P4Obrazce

```
class Ctverec:
    def __init__(self, strana):     # __konstruktor__,  explicitne uvadeny  self  = konvence pro this
        self.strana = strana        # přiřazení = i definici lokální proměnné každé instance
    def vypocitejPlochu(self):
        return self.strana**2

class Kruh:
    def __init__(self, polomer):
        self.polomer = polomer
    def vypocitejPlochu(self):
        import math
        return math.pi*(self.polomer**2)

class Obdelnik2x3:  # kdyz nema konstruktor zadny parametr, nemusi se uvest
    def vypocitejPlochu(self):
        return 2*3

seznam = [Kruh(8), Ctverec(2.5), Kruh(3), Obdelnik2x3()]

for tvar in seznam:
    print "Plocha je: ", tvar.vypocitejPlochu() # tady se jiz self neuvádí, není zde ani definované
```

Python třídy a objekty

Dědit lze i od rodiče z jiného modulu

```
class Potomek(modul.Rodic):
        <příkaz1>
        . . .
         <příkazN>
```

Příkazy jsou nejčastěji definicemi metod. Lze definovat vnořené třídy

Metody jsou implicitně dynamické (virtuální) a mohou překrýt metodu rodiče

Statickou lze metodu udělat i po definici jejím zasláním metodě *staticmethod*

```
jménometody = staticmethod(jménometody)
```

Podobné jsou metody třídy, nemají také self parametr

```
jménometody = classmethod(jménometody)
```

Lze definovat destruktor __del__, ten se provede (dělá úklidové akce), když je objekt rušen, např. výstup z rozsahu platnosti objektu.

Python třídy a objekty

Má násobnou dědičnost

```
class Potomek(Rodic1, Rodic2, …RodicM):
        <příkaz1>
        . . .
         <příkazN>
```

Řešení problému násobného dědění

Není-li atribut v Potomkovi, hledá se v Rodiči1, pak v Rodiči Rodiče1, …v Rodiči2…

Tj. do hloubky a pak z leva do prava .

Př. P42.py

Python třídy a objekty

```
class Otec:
      def __init__(self):
        self.oci = 'zelene'
        self.usi = 'velke'
        self.ruce = 'sikovne'
    def popis(self):
        print self.oci, self.usi, self.ruce

class Matka:
    def __init__(self):
        self.oci = 'modre'
        self.nos = 'maly'
        self.nohy = 'dlouhe'
    def popis(self):
        print self.oci, self.nos, self.nohy

class Potomek(Matka, Otec):
    def __init__(self):
        Otec.__init__(self)        # 1.
        Matka.__init__(self)       # 2.
    def popis(self):
        print self.oci, self.usi, self.ruce, self.nos, self.nohy

petr = Potomek()
petr.popis()
```

Python třídy a objekty

Spustit bud v Idle nebo v d:\pgs\python>c:\sw\Python25\python a pak >>>import P42

Co vypíše?

modre velke sikovne maly dlouhe

Při změně na

```
class Potomek(Matka, Otec):
    def __init__(self):
        Matka.__init__(self)
        Otec.__init__(self)
```

Vypíše

zelene velke sikovne maly dlouhe

Př P41perzistence.py

Ilustruje zcela obecnou možnost vytváření perzistentních objektů = uložení objektu do souboru (to funguje v každém jazyce)

Python perzistentní objekty

```
class A:                              # příklad P41perzistence.py
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def save(self, fn):               # uložení objektu do souboru
        f = open(fn, "w")
        f.write(str(self.x) + '\n') # převeď na řetězec
        f.write(str(self.y) + '\n')
        return f    # do stejného souboru budou své hodnoty
                    # připisovat objekty odvozených tříd
    def restore(self, fn):            #  obnovení objektu ze souboru
        f = open(fn)
        self.x = int(f.readline()) # převeď zpět na původní typ
        self.y = int(f.readline())
        return f
a = A(1, 2)                           # Vytvorime instance.
a.save('a.txt').close()               # Ulozime instance.
newA = A(5, 6)                        # Obnovime instance.
newA.restore('a.txt').close()
print "A: ", newA.x, newA.y      # Podivej se na disk, je tam soubor a.txt a v nem 1, 2
```

Python třídy a objekty

Python má i destruktor

```
def __del__(self):
    <příkazy>  # píkazy se provedou když je objekt odstraňován z paměti
               # to nastane samovolně, když čítač odkazů na objekt je 0 (objekt má čítač),
               # takže destruktor se moc nepoužívá (oproti C++ není tak důležitý)
```

Př.P43Destruktor.py

Po dokončení fce *zkus( )* bude počet odkazů na objekt *logickeJmenoSouboru* nula, takže překladač automaticky zavolá definovaný destruktor

Př. P43Destruktor.py

```
class UkazkaDestruktoru:
    def __init__(self, soubor):            #vytvori soubor, otevre ho a zapise do nej
        self.file = open(soubor, 'w')
        self.file.write('tohle zapisuji do souboru\n')
    def write(self, retezec):
        self.file.write(retezec)
    def __del__(self):  # destruktor, write overime ze se provedl
        self.write( "__del__ se provedlo")

def zkus():
    logickeJmenoSouboru = UkazkaDestruktoru('pomocnySoubor')
    logickeJmenoSouboru.write('tohle take zapisuji do souboru\n')
    # zde objekt logickeJmenoSouboru prestane existovat

zkus()
```

Po spuštění se podívej na soubor pomocnySoubor

Python třídy a objekty

Privátní atributy

Omezená podpora formou: ***__jméno*** *je textově nahrazeno* ***__classname__jméno***
a tímto jménem je atribut nadále přístupný

Prázdné třídy poslouží jako typ záznam např.

```
class Record:
    pass

petr = Record( )   # vytvoří prázdný záznam o Petrovi

# jednotlivé položky instance není nutné deklarovat předem, lze to provést dodatečně
petr.jmeno = 'Petr Veliky'
petr.vek = 39
petr.plat= 40000
```

Python – výjimky jsou také třídy

Obecný tvar pythonských výjimek:

```
try:
    # ošetřované příkazy
except TypVyjimky:
    # zpracování výjimky
except TypVyjimky:   # při neuvedení TypVyjimky zachytí se všechny dosud nechycené
    # zpracování výjimky
. . .
else:
    # činnost, když nennastane výjimka (nepovinné)

finally:
    # činnost prováděná ať výjimka nastane či nenastane
```

Vyhození výjimky způsobíme příkazem **raise JmenoVyjimky** # = instance výjimky

Možné formy: raise TřídaTypuException, instance
raise instance je zkrácením raise instance._class_, instance
samotné raise lze užít jen uvnitř except a vyhodí posledně nastalou výj.

Python – výjimky jsou také třídy

Př. P44vyjimky.py Čte řádek souboru, třetí údaj je považován za číslo. Je-i 122 způsobí dělení 0 výjimku, při všech jiných vadách zachytává nespecifikovanou výjimku

```
print 'Start programu.'
try:
    data = file(‘data.txt')
    print 'Soubor s daty byl otevren.'
    try:
        hodnota = int(data.readline().split()[2])  #  radek tvaru slovo slovo cislo …
        print 'Hodnota je %d.' % (hodnota/(122-hodnota))
    except ZeroDivisionError:
        print 'Byla nactena hodnota 122.'
    except: print "Stalo se neco neocekavaneho."
finally:
    data.close()
    print 'Soubor s daty byl uzavren.'
print 'Konec programu.'
```

Python – výjimky jsou také třídy

Uživatelem definované výjimky jsou instance tříd odvozených z třídy Exception.
Hierarchie zpracování děděných výjimek je jako v Javě

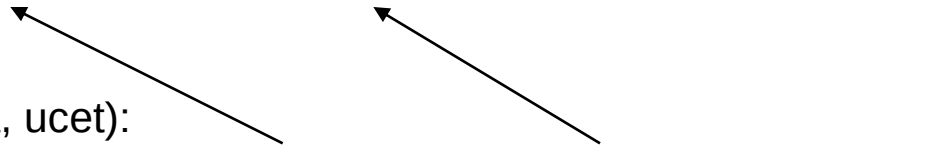
Př.P45.py

```
class NejakaError(Exception):
    pass
try:
    raise NejakaError , ' Informace co se deje \n‘ #zpusobime vyjimku
except NejakaError:
    print u'Narazili jsme na chybu při zpracování.'  #zpracovani vyjimky
```

Př.P46.py Přetažení konta

```
# -*- coding: cp1250 -*-
class ChybaZustatku(Exception):
    hodnota = 'Nelze vybrat. Na vašem účtu je jen %5.2f korun.'
class Ucet:
    def __init__(self, pocatecniVklad):
        self.stav = pocatecniVklad
        print 'Mate založen účet s vkladem  %5.2f korun.' % self.stav
    def vlozit(self, castka):
            self.stav = self.stav + castka
    def vybrat(self, castka):
        if self.stav >= castka:
            self.stav = self.stav - castka
        else:
            raise ChybaZustatku, ChybaZustatku.hodnota % self.stav
    def zustatek(self):
        return self.stav
    def prevod(self, castka, ucet):
        try:
            self.vybrat(castka)
            ucet.vlozit(castka)
        except ChybaZustatku, e:
            print str(e)

mujUcet = Ucet(300)
mujUcet.vybrat(200)
mujUcet.vybrat(200)             # lze vyzkouset ruzne castky
```



Python – událostmi řízené programování a GUI

Vlastnosti:

- **Program po spuštění čeká v nekonečné smyčce na výskyty událostí**
- **Při výskytu události provede odpovídající akci a nadále čeká ve smyčce**
- **Skončí až nastane konec indikující událost**

Události může generovat OS (obvyklé u programů s GUI), nebo vnější čidla

Předvedeme na freewarovém multiplatformním systému Tk (Tkinter pro Python)

Př. P5udalosti.py

Program zachycující události stisknutí klávesy, dokud nenastane ukončující událost (stisk mezery). Na stisk klávesy program reaguje výpisem kódu klávesy

Program používá modul Tkinter s prostředky pro GUI

1. Vytvoříme třídu KlavesovaAplikace pro naši aplikaci
2. Tato třída obsahuje metody pro zpracování událostí
3. Součástí konstruktoru třídy je vytvoření GUI okna pro výpis kódu klávesy
4. Vytvoříme instanci třídy
5. Této instanci zašleme zprávu *mainloop*

!!! Ale pozor, spouštěj ho z příkazového řádku OS, … c:\sw\Python25>python ne z IDLE, protože IDLE samo využívá Tkinter

>>> import P5udalosti

Nebo poklepem na ikonu souboru P5udalosti.py

```
from Tkinter import *
class KlavesovaAplikace(Frame):          # Vytvori GUI
    def __init__(self):
        Frame.__init__(self)        #vytvori ramec do ktereho budou vkladany dalsi prvky
        self.txtBox = Text(self)    #vytvori a vlozi do ramce prvek pro praci s radky textu
        self.txtBox.bind("<space>", self.zpracujUdalostUkonceni) #navaze mezeru na udalost
        self.txtBox.pack()          #pack je manazer umisteni. Vlozi prvek textovy box do jeho rodice
        self.pack()                 #az ted se textovy box zviditelni
        self.txtBox.bind("<Key>", self.zpracujStiskKlavesy) #navaze stisk klavesy na zpracujStiskKl…
    def zpracujUdalostUkonceni(self, udalost):
        import sys
        sys.exit()
    def zpracujStiskKlavesy(self, udalost):              # metoda zpracovani udalosti
        str = "%d\n" % udalost.keycode      #str nabyde hodnotu kodu klanesy
        self.txtBox.insert(END, str)        #vlozi text za posledni znak
        if udalost.keycode == 88:           # to je x
            self.txtBox.insert(END, 'cteme x') #vlozi text za posledni znak
        return "break"
mojeAplikace = KlavesovaAplikace()      # vytvoreni instance
mojeAplikace.mainloop()                 #spustime cekaci smycku
```

Python – událostmi řízené programování a GUI

Některé prvky GUI Tkinter, lze vyzkoušet interaktivně

>>> **from Tkinter import *** naimportuje jména ovládacích prvků

>>> **top = Tk( )** vytvoří **widget (ovládací prvek)** na nejvyšší úrovni, ostatní budou jeho potomky. Je to okno, do něj budeme další prvky přidávat

>>> **dir(top)** vypíše všechna jména = atributy objektu top třídy Tk

>>> **F = Frame(top)** vytvoří widget rámec (frame), který je potomkem top. Do něj budou umisťovány ostatní ovládací prvky. Objekt z Frame má již atribut pack Funguje i F=Frame() t.j. primo bez zavedeni top

>>> **F.pack( )** aktivuje packer, protoze je ramec zatim prazdny, zdrcne ho na listu

>>> **lPozdrav = Label(F, text = 'everybody out')** vytvoří objekt třídy Label jako potomka F, jeho atribut text má iniciovanou hodnotu. Lze udat i barvu a typ písma …

>>> **lPozdrav.pack( )** pakuje lPozdrav do rámce, ted se objevi v okenku

>>> lPozdrav.configure(text = 'vsichni ven') metoda configure dovoluje změnit vlastnosti objektu takze zde zameni text za novy

>>> lPozdrav['text'] = "everybody out" menime-li jen jednu vlastnost, tak je tohle kratší

Python – událostmi řízené programování a GUI

>>> F.master.title(‘Ahoj’) dovoluje nastavit titulek okna metodou title pro widget na vrcholu hierarchie = objekt top

>>> bQuit = Button(F, text= ‘Konec’, command = F.quit) vytvoří tlačítko s nápisem Konec, které je spojeno s příkazem F.quit. Předáváme jméno metody quit

>>> bQuit.pack( ) zajistí zviditelnění tlačítka

>>> top.mainloop( ) odstartuje provádění smyčky. Činnost teď řídí Tkinter. Zmizely **>>>** a objeví se až po stisku tlačítka Konec a pak lze provest >>> quit()

Př. P51

Je skriptem, který to dělá.

Musí se spustit z příkazové řádky OS import P51 nebo poklepem na ikonu

Vzniklé okno čeká ve smyčce na stisk Konec …

Python – událostmi řízené programování a GUI

```
from Tkinter import *       #naimportuje jména ovládacích prvků

# Vytvorime okno.
top = Tk()                  # vytvoří widget (ovládací prvek) na nejvyšší úrovni
F = Frame(top)              # vytvoří widget rámec (frame), který je potomkem top
F.pack()                    # pakuje rámec na lištu

# Pridame ovladaci prvky.
# vytvoří objekt třídy Label jako potomka F
lPozdrav = Label(F, text="everybody out")
lPozdrav.pack()             # pakuje lPozdrav do rámce
F.master.title('Nazev')     # nastavi titulek okenka
# vytvoří tlačítko s nápisem Konec, barvy, specifikuje příkaz asociovaný s uvolň. tlačítka
bQuit = Button(F, text="Konec", fg='white', bg='blue', command=F.quit)
bQuit.pack()                # zajistí zviditelnění tlačítka (umístí ho do rámce)
# Spustime smycku udalosti.
top.mainloop()
quit()                       # stisk Konec ukonci vypocet
```

Python – událostmi řízené programování a GUI

Př. P52.py reakce na tlacitka a na souradnice klicku

```
# -*- coding: cp1250 -*-        #jsou tam ceska pismena
from Tkinter import *
vrchol = Tk()                   #vytvori ovladaci prvrk na nejvyssi urovni

def odezva(e):                  # definuje reakci na udalost odezva
   print 'klik na', e.x, e.y    # tiskne souradnice x, y
def klik():                     #definuje odezvu na stisk tlacitka
    print u"Stiskl jsi tlačítko!"

f = Frame(vrchol, width=200, height=300)        #parametry urci velikost ramce
# pružněji spoji ramec f s levym tlacitkem mysi a odezvou metoda bind
f.bind('<Button-1>', odezva)  #<Button-1> je leve, <Button-2> je prave tlacitko
f.pack()
for i in range(4):
    tlacitko=Button(text=u"Já jsem tlačítko", command=klik)
    tlacitko.pack()

vrchol.mainloop()
quit()   #ukonci beh Pythonu po zavreni okna
```

Př.53 Okno se zapisovacím polem, horkou klávesou a tlačítky mazání a konec

```
# -*- coding: cp1250 -*-
from Tkinter import *

# Nejdříve vytvoříme funkci pro ošetření události.
def vymazat():
  eTxt.delete(0, END)    # metoda maze text od nulteho znaku do konce

def eHotKey(u):
  vymazat() # zavedeme pro mazani i hot key

# Vytvoříme hierarchicky nejvyšší okno a rámeček.
hlavni = Tk()
F = Frame(hlavni)
F.pack()
```

```
# Nyní vytvoříme rámeček s polem pro vstup textu.
fVstup = Frame(F, border=20) # velikost okna je s parametrem 20
eTxt = Entry(fVstup) # prvek tridy Entry je pro zadavani jednoradkoveho textu
eTxt.bind('<Control-m>', eHotKey) # navazani ctrl-m na mazani
fVstup.pack()
eTxt.pack()

# Nakonec vytvoříme rámečky s tlačítky.
# Pro zviditelnění je vmáčknutý = SUNKEN
fTlacitka = Frame(F, relief=SUNKEN, border=1)
bVymazat = Button(fTlacitka, text="Vymaz text", command=vymazat)
bVymazat.pack(side=RIGHT, padx=5, pady=2) #5 a2 jsou vycpavky (mista) mezi prvky
bKonec = Button(fTlacitka, text="Konec", command=F.quit)
bKonec.pack(side=LEFT, padx=5, pady=2)
fTlacitka.pack(side=TOP, expand=True)

# Nyní spustíme čekací smyčku
F.mainloop()
quit()
```

Python – událostmi řízené programování a GUI

Př.P54.py OO přístup ke GUI aplikacím
Celá aplikace se zapouzdří do třídy buď tak, že

1) Odvodíme třídu aplikace od tkinter třídy Frame (užívá dědičnost) nebo
2) Uložíme referenci na hierarchicky nejvyšší okno do členské proměnné (užívá kompozici)

Použijeme postup 2 pro konstrukci s polem typu Entry, a tlačítky Vymaz a Konec v OO podobě.

- Do konstruktoru aplikace dáme jednotlivé části GUI.
- Referenci na prvek typu Frame přiřadíme do self.hlavniOkno, čímž zajistíme metodám třídy přístup k prvku typu Frame
- Ostatní prvky, ke kterým přistupujeme přiřadíme členským proměnným instance z Frame
- Funkce pro zpracování událostí se stanou metodami třídy aplikace, takže mohou přistupovat k datovým členům aplikace pomocí reference self.

```
# -*- coding: cp1250 -*-
from Tkinter import *

class AplikaceVymazat:
    def __init__(self, rodic=0):
        self.hlavniOkno = Frame(rodic,width=200,height=100)
        self.hlavniOkno.pack_propagate(0) #aby platila zadana vyska, sirka a ne implicitni
        # Vytvoříme widget třídy Entry
        self.vstup = Entry(self.hlavniOkno)
        self.vstup.insert(0, "Pocatecni text")
        self.vstup.pack(fill=X) #prvek vstup zabira ve smeru X cele mozne misto
```

```
# Nyní přidáme dvě tlačítka v ramecku a použijeme efekt drážky.
    fTlacitka = Frame(self.hlavniOkno, border=2, relief=GROOVE) #drazkovany relief
    bVymazat = Button(fTlacitka, text="Vymazat",
                      width=8, height=1, command=self.vymazatText)
    bKonec = Button(fTlacitka, text="Konec",
                    width=8, height=1, command=self.hlavniOkno.quit)#velikost tlacitka
    bVymazat.pack(side=LEFT, padx=15, pady=1) # urcuji stranu a vnejsi vzdalenosti
    bKonec.pack(side=RIGHT, padx=15, pady=1)
    fTlacitka.pack(fill=X)   #vyplneni tlacitek ve smeru X
    self.hlavniOkno.pack()

    # Nastavíme nadpis okna.
    self.hlavniOkno.master.title("Vymazat")

  def vymazatText(self):
    self.vstup.delete(0, END)        #od 0 do konce vymazat

aplikace = AplikaceVymazat()
aplikace.hlavniOkno.mainloop()
quit()
```

Python – událostmi řízené programování a GUI

Běžné Tk prvky:

Tlačítko	**Button**
Plátno	**Canvas**
Zaškrtávací tlačítko	**Checkbutton**
Jednořádkový vstup	**Entry**
Rámeček	**Frame**
Více řádek textu	**Listbox**
Nálepka	**Label**
Rolovací menu	**Menu**
Tlačítko výběru	**Menubutton**
Radiové tlačítko	**Radiobutton**
Posuvný ovladač	**Scale**
Posuvný ukazatel	**Scrollbar**
Textový editor	**Text**

Python a databáze

Nainstalovat pyodbc takto: **(pyodbc = object database connectivity)**

- **Ovládací panely**
  - **Nástroje pro správu**
    - **Datové ODBC zdroje**
      - **Přidat**
        - » **Vybrat databázi (driver do MS Access)**
        - » **Dokončit**
        - » **Název zdroje dat (třeba Lidé, je to název propojení)**
        - » **Vybrat**
        - » **Na C:/ je v MS Access vytvořená databáze1.mdb, tak ji vyber**
        - » **OK**
      - **Objeví se v Správce zdrojů dat ODBC**
      - **OK**
- **Zavřít okno**

Python a databáze

Relační databázový model: tabulky, atributy, n-tice (záznamy)

Tabulka Studenti

kod	Jmeno	Prijmeni	Vek	Škola	Prumer
12	Karel	Dadak	23	VUT	2,01
07	Jana	Tlusta	19	CVUT	1,95
28	Josef	Hora	20	ZCU	1,75
…	…	…	…	…	…

Primární klíč = jeden nebo více atributů, sloužící k jednoznačné identifikaci záznamu. Např. jméno a příjmení, nebo rodné číslo, apod.

Databázi chápejme jako soustavu tabulek navzájem propojených přes společné atributy

Python a databáze

Jazyk SQL:

SELECT	výběr hodnot specifikovaných atributů
FROM	určení tabulek ze kterých se dělá select nebo delete
INNER JOIN	spojení záznamů z více tabulek k získání jediné výsl. relace
WHERE	určení podmínek, které musí splňovat vybírané záznamy
GROUP BY	určení podmínek pro seskupování vybíraných záznamů
ORDER BY	určení podmínek dle kterých se uspořádají vybírané záznamy
INSERT	**INTO** tabulku (atributy) **VALUES** (hodnoty)
UPDATE	tabulku **SET** atribut **=** hodnota, atd **WHERE** podminka
DELETE	**FROM** tabulka **WHERE** podminka

Python a databáze

Př.

select Jmeno, Prijmeni from Studenti where Vek > 23

select Jmeno, Prijmeni, Prumer from Studenti where Vek > 23 order by Prumer, Vek

select * from Studenti

insert into Studenti (Jmeno, Prijmeni, Vek, Skola, Prumer)
values (‘Jan’, ‘Novy’, 25, ‘CVUT’, 3,1)

delete from Studenti where Vek>39 and Prumer>3

update Studenti set Skola=‘VUT’, Vek= 12 where Jmeno=‘Gustav’ and Prijmeni=‘Klaus’

Často je třeba vyhledat podmnožinu kartézského součinu více tabulek
Např tabulek Studenti a Skoly, kde Skoly vyjadřuje relaci Škola a Město kde sídlí

Select IDcislo, prumer from Studenti inner join Skoly on Studenti.Skola = Skoly.Skola
where Skola.Mesto = Praha order by IDcislo

Python a databáze

DB-API zajistí propojení s databází, vytvoří a zviditelní objekt **kurzor** dovolující provádět operace na databázi (selekty, inserty, updaty, delety). V objektu kurzor jsou interně uloženy výsledky dotazu

K výběru řádků výsledku dotazu v podobě objektu lze použít metody:

fetchone() vrací n-tici = další řádek výsledku uloženého v kurzoru

fetchmany(n) vrací n řádků, které jsou na řadě ve výsledku uloženém v kurzoru

fetchall() vrací všechny řádky výsledku.

Současně dochází k posouvání kurzoru

Python používá SQL jako vnořený jazyk, doplní ho na úplný jazyk.

commit příkaz k zakončení transakce (zapsání změn do databáze)

close uzavření kurzoru, uzavření spojeni

Python a databáze

Spustit Python, třeba Idle

```
>>> import pyodbc
>>> c=pyodbc.connect("DSN=Lide")          #vytvori spojeni c se zdrojem dat (byl pojmenovan Lide)
>>> cu=c.cursor()                         #vytvori kurzor cu
>>> cu.execute("select Jmeno, Prijmeni, Vek, Skola from Studenti")
<pyodbc.Cursor object at 0x012C32A0>
>>> for row in cu: print row.Jmeno, row.Prijmeni,row.Skola
Vypíše se výsledek a kurzor cu se dostane na konec (musí se znovu nastavit příkazem
    cu.execute)
>>> cu.execute('select sum(vek),prumer from Studenti group by prumer')
<pyodbc.Cursor object at 0x00DA33E0>
>>> for r in cu: print r[0],r[1]
Vypíše dvojice (suma veku, prumer) tj. ti se stejným průměrem budou agregováni na 1 řádek
>>> cu.execute("update Studenti set Skola ='TUO' where Skola = 'VSB'")
číslo                                  #vypisuje počet zasažených řádků
>>> cu.execute("delete from Studenti where Skola = 'TUO'")
Číslo                                  #vypisuje počet zasažených řádků
>>> cu.execute("insert into Studenti (Jmeno, Prijmeni, Vek, Skola, Prumer) values ('Krystof',
    'Kolumbus', 450, 'Zivota', 1.1)")
1
>>> c.commit()                         #musí se udelat, není zde autocommit
>>> c.close()                                  #uzavre spojeni
```

Python a databáze

Výběr dat z více tabulek – inner join

Tabulka Studenti viz dříve

Tabulka Skoly se sloupci

Škola	Město
ZCU	Plzen
KU	Praha
CVUT	Praha
VUT	Brno
VSB	Ostrava

Dotaz na studenty z Plzně

```
>>> cu.execute("select Prijmeni, Vek from Studenti inner join Skoly on
    Skoly.Skola=Studenti.Skola where Skoly.Mesto='Plzen'")

>>> for row in cu: print row.Prijmeni,row.Vek
```

```
>>> cu.execute("select * from Studenti where Vek>10")
<pyodbc.Cursor object at 0x011CB110>
>>> vsechnaPole = cu.description          #metoda vybere zahlavi vysledku
>>> for pole in vsechnaPole: print pole[0] #pole[0]= jmeno sloupce zahlavi

kod
Vek
Jmeno
Prijmeni
Skola
Prumer
>>> for pole in vsechnaPole: print pole[1] # pole[1] =typ sloupce zahlavi

<type 'int'>
<type 'float'>
<type 'str'>
<type 'str'>
<type 'str'>
<type 'float'>
```

```
>>> vsechnyZaznamy = cu.fetchall() #vyber  vsech zaznamu z kurzoru
>>> for zaznam in vsechnyZaznamy:
…           print "\n"
…           for polozku in zaznam:
…                       print polozku
…
1 12.0 Karel Dadak VUT 2.00999999046
2 9.0   Jana  Tlusta CVUT 1.95000000021
3 23.0 Josef Hora ZCU 1.45000004768
4 56.0 Krystof Kolumbus KU 1.29999995232
5 31.0 Josef Druhy ZCU 2.04999995232
>>> for row in cu: print row.Prijmeni,row.Vek
    //nic se nevytiskne, protože kurzor se provedením fetchall dostal az na konec
>>> cu.execute("select * from Studenti where Vek>10")   nově ho naplníme
>>> jedenZaznam = cu.fetchone()
>>> print jedenZaznam[3]
Dadak
>>> jedenZaznam = cu.fetchone()
>>> print jedenZaznam[3]
Hora
```

Pozor, kurzor dává interní reprezentaci objektu, pro výběr hodnot nutno použít index

for zaznam in vsechnyZaznamy: print zaznam

<pyodbc.Row object at 0x00A5A368>
<pyodbc.Row object at 0x00A5A4E8>
<pyodbc.Row object at 0x00A5A770>
<pyodbc.Row object at 0x00A5A140>
>>> for zaznam in vsechnyZaznamy: print zaznam[2],

Karel Josef Krystof Josef
>>> for zaznam in vsechnyZaznamy: print zaznam[-1],

2.00999999046 1.45000004768 1.29999995232 2.04999995232