4. Export dat pomocí Pythonu

V souvislosti s V4 technologií velmi výrazně vzrostla důležitost jazyka Python ve všech produktech Vemy. Mnohé interní procesy, konverzní moduly, funkce ve Správci apod. jsou přepisovány právě do tohoto jazyka, který je na rozdíl od stávajícího C++ na tyto činnosti velmi vhodný vzhledem ke své relativní jednoduchosti a flexibilitě.

Účelem tohoto textu a přednášky je na jednoduchém a konkrétním případě ukázat využití jazyka Python zákazníkem a to na nejčastěji poptávaném případě – exportu a případně importu dat z/do databáze Vema. Hlavní motivací bývá potřeba zákazníků vytvářet různé reporty, napojit data z Vemy do jejich dalších systémů (obědy, ovládání vjezdové brány) apod.

Společnost Vema nabízí celou řadu modulů, které se dají použít na export dat do různých systémů (v současné době se jedná o 56 modulů, které obsahují v názvu export). Tyto moduly jsou pravidelně aktualizovány a je u nich garantována funkčnost a podpora i do budoucna. V případě vlastních skriptů tomu tak není a zákazník je nucen pravidelně sledovat, zda-li jím vytvořený skript dělá stále to co dělat má a případně si ho opravit.

4.1 Python

Jedná se o objektově orientovaný interpretovaný programovací jazyk.

- ***Objektově orientovaný*** – znamená, že Python rozumí třídám, objektům, metodám, atributům, dědění apod. Tedy programátorským technikám, které dovedou velmi zpřehlednit vlastní program, soustředit se na podstatné věci a neřešit něco, co vyřešil již někdo jiný – např. nízkoúrovňový přístup do Vema databáze. V Pythonu je navíc objektem všechno, byť to není na první pohled patrné.
- **Interpretovaný** – příkazy jazyka Python jsou zapsány do textového souboru – skriptu. Skript se pak musí spustit pomocí tzv. interpretu (programové vybavení, které umožňuje skripty v jazyce Python spouštět – python.exe). Výsledkem tedy není žádné exe.

  Příklad spuštění skriptu pokus.py: `python.exe pokus.py`

- ***Interpret*** - Nejsnazší cestou k získání interpretu je instalace balíku VemaPython. V současné době by se měl vyskytovat již na všech instalacích V4 serveru, neboť se jedná o povinnou systémovou součást. Balík VemaPython obsahuje Python verze 2.6.x a také všechny potřebné třídy pro práci s DB Vema. Alternativně lze interpret stáhnout na webu www.python.org, ale po instalaci budou chybět zmíněné třídy. Doporučuje se vyhnout verzím 3.x, neboť pro ně není na straně Vemy (nejen) prozatím podpora.

4.2 Nejzákladnější vlastnosti jazyka Python

V následujícím odstavci jsou zmíněny nejzákladnější charakteristiky jazyka Python a to především z důvodu, aby nebylo nutno studovat externí zdroje a přitom byla syntaxe popsaných skriptů rámcově pochopitelná.

Skript obvykle začíná definicí znakové sady, ve které bude skript napsán – v případě Vemy utf-8.

```
# -*- coding: utf-8 -*-
```

Komentáře, resp. dočasné zneplatnění řádku se provede znakem #

```
print "Ahoj světe"          #tento řádek bude proveden
#print "Ahoj světe"         #tento řádek nebude proveden
```

Proměnné se nemusí dopředu deklarovat - tzn. není potřeba na začátku skriptu psát, že do proměnné a se bude ukládat číslo a do proměnné b pak řetězec. Čili je možné kdykoliv uprostřed skriptu začít používat libovolnou proměnnou s libovolným obsahem.

```
a = 5
mojeNovaPrommenna = "Ahoj Svete"
jinyUTFRetezezec = u'Příliš žluťoučký kůň úpěl ďábelské ódy'
```

Pozor je však nutné dát na to, aby před použitím v nějaké operaci byla proměnná inicializovaná. Následující příklad skončí s chybou, neboť proměnná c nebyla inicializována – tzn. neexistuje.

```
x = 2 * c
```

Správného výsledku dosáhneme takto:

```
c = 3
x = 2 * c
```

Veškeré názvy proměnných (funkcí, tříd..) jsou citlivé na malá/velká písmena. `Vysledek` a `vysledek` jsou dvě zcela odlišné proměnné – Python je tedy case-sensitive jazyk.

```
Vysledek = True
vysledek = False
```

Asi nejvýraznější vlastností Pythonu ve srovnání s jinými běžnými jazyky je způsob členění bloků. Bloky slouží k seskupení příkazů, například uvnitř cyklů, funkcí, objektů, struktur atd.

Například v Pascalu jsou bloky označeny slovy BEGIN a END a v jazyce C slouží pro vytváření bloků špičaté závorky {}. V Pythonu se pro vytváření bloků používá odsazování.

Nový blok se vytvoří tak, že se napíše na začátky řádků před příkazy, které spolu tvoří blok, libovolný počet mezer nebo tabulátorů. Ale před každým dalším příkazem v bloku musí být stejný počet mezer a tabulátorů. Odsadí-li se o mezeru více, nebo pokud se místo tabulátoru použijí mezery, začne se tím jiný blok. Pokud se vytvoří nový blok na místě, kde nemá být, skončí program s chybou – IndentationError.

Příkladem budiž podmínka, která se rozhoduje dle hodnoty proměnné a (vlevo Python, vpravo C):

```
a=5
if a<5:
  print "a je mensi jak 5"
else:
  print "a je vetsi nebo rovna 5"
print "hotovo"
```

```
int a=5;
if(a<5) {
  printf("a je mensi jak 5");
} else {
  printf("a je vetsi nebo rovna 5");
}
printf("hotovo");
```

Na prvním řádku vložíme do proměnné a číslo 5. Následuje rozhodovací podmínka. V Pythonu není potřeba psát v příkazu if závorky – tedy za předpokladu, že je podmínka jednoduchá a nehrozí chybné vyhodnocení dle odlišné priority operátorů. Za podmínkou následuje dvojtečka, která Pythonu říká, že na dalším řádku bude následovat nový blok, tedy sada příkazů, které budou zleva odsazeny o stejný počet mezer nebo tabulátorů. V tomto případě je odsazení o dvě mezery. Následuje else, další dvojtečka a další blok odsazený o 2 mezery. Nakonec tento skript vytiskne „hotovo".

Obdobně funguje hlubší zanoření, které je zde ve formě ukázky definice funkce:

```
def Porovnej(a, b):
  vysledek = "cisla jsou shodna"
  if a>b:
    vysledek="a je vetsi jak b"
  if b<a:
    vysledek="b je vetsi jak a"
  return vysledek

print Porovnej(3,5)
```

První řádek definuje funkci Porovnej se dvěma parametry a, b. Všechny následující příkazy patřící dané funkci musí být odsazeny. To platí i pro odsazení podmínek v dané funkci.

Z uvedeného je zřejmé, že striktní dodržování odsazení přispívá k zachování přehlednosti skriptu. Programátor je tak nucen dodržovat určitou „štábní" kulturu zdrojového kódu, která bývá u jiných jazyků často problematická.

4.2.1 Datové typy seznam (list) a slovník (dictonary)

Datový typ seznam je něco jako pole v ostatních jazycích. Jedná se o proměnnou, která v sobě může uchovávat jednu, dvě nebo klidně milion hodnot. Hodnoty jsou indexovány od nuly.

```
mujSeznam = ['PAM', 'PER', 'ELD', 'RNP']
mujSeznam.append('KZP')
print mujSeznam[1]
```

První řádek založí proměnnou mujSeznam typu seznam a naplní ho zadanými prvky. Druhý řádek přidá do seznamu mujSeznam nový prvek KZP. Třetí řádek pak vytiskne prvek s indexem 1, tedy PER.

Datový typ slovník je obdoba datového typu seznam s tím rozdílem, že jednotlivé prvky nejsou indexovány, ale vytvořeny dvojicí klíč-hodnota. Unikátní klíč tedy identifikuje konkrétní hodnotu ve slovníku.

```
mujSlovnik = {'PAM': 'Mzdy', 'PER': 'Personalistika', 'ELD': 'El. podani ELDP'}
mujSlovnk['RNP'] = 'Registrace nemocenskeho pojisteni'
print mujSlovnik['PER']
```

První řádek založí novou proměnnou typu slovník a naplní ji třemi dvojicemi klíč-hodnota. Druhý řádek přidá do slovníku další dvojici klíč-hodnota. Třetí řádek poté vytiskne hodnotu prvku s klíčem PER, tedy Personalistika.

4.3 Exportní skript

Vlastnosti:

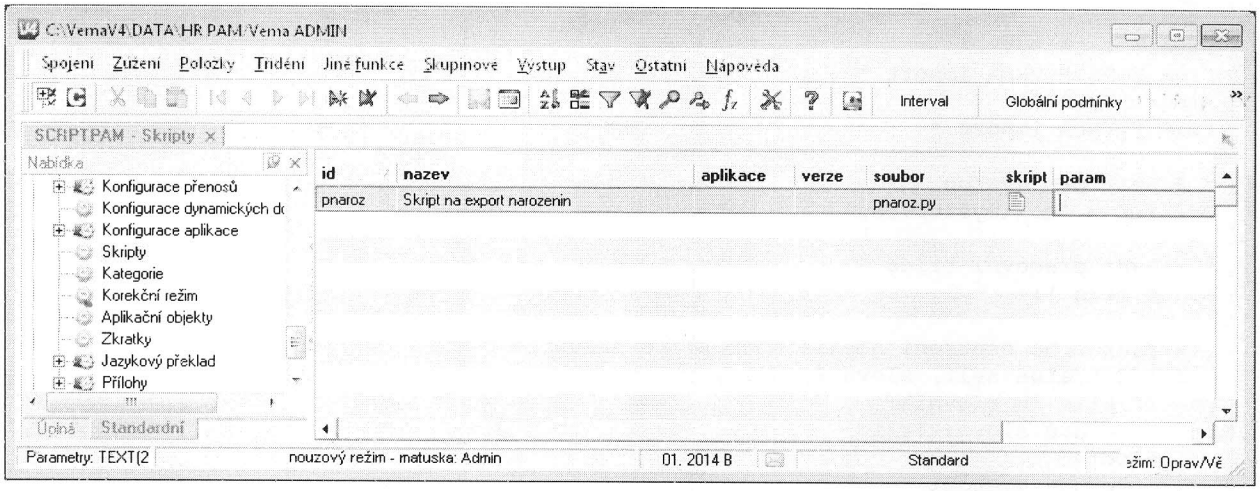
- Skript bude číst obsah souboru F1 v PAM.
- Výstupem bude CSV soubor se seznamem pracovníků s platným PPV a daty jejich narození.
- Spouštěn bude pomocí parametru DBVCommand Vema klienta.

4.3.1 Spouštění

V úvodu těchto textů bylo uvedeno, že Python je jazyk interpretovaný a že se musí spouštět pomocí interpretu python.exe. To je samozřejmě pravda, avšak Vema skýtá ještě jednu možnost spouštění a tím je zapsání existujícího skriptu do souboru Skripty v jakékoliv DBV aplikaci a spouštění zavoláním Vema klienta s parametrem DBVCommand. Výhodou tohoto způsobu spouštění je, že se skript bude spouštět vždy s poslední verzí Pythonu dostupného v balíku VemaPyhton a že není potřeba řešit přímo ve skriptu připojování k V4 serveru – připojení se převezme přímo ze spuštěné aplikace. Třetí výhodou může být možnost spuštění pomocí klávesy F7 přímo v DBV aplikaci.

Nastavení v DBV aplikaci (PAM):

- Otevřít soubor skripty
- Založit novou větu, kde:
  - *id* je libovolný identifikátor skriptu (bude použit v DBVCommand)
  - *nazev* je textový popis skriptu
  - *soubor* je název fyzického souboru se skriptem v MOD adresáři



Spuštění se provede následující dávkou (vše na jednom řádku):

```
Vema.exe /Role="Admin" /App=DBV /DataName=HR /Subsystem=PAM /DBVCommand="standard
71 pnaroz" /wait
```

- standard **71** pnaroz – zobrazí okno s výstupem skriptu a po dokončení se zavře
- standard **70** pnaroz – zobrazí okno s výstupem skriptu a po dokončení čeká na interaktivní potvrzení. Pro dávkový a automatizovaný režim je toto nevhodné.

4.3.2 Skript pnaroz.py

Soubor pnaroz.py je umístěn v MOD adresáři a jeho obsah je následující:

```
# -*- coding: utf-8 -*-

import sys
import os
import VmDevPyVRDA
from VmDevPyVRDA import *
import PyVRDA
from PyVRDA import *

# Soubor pro vystup
OUTFILE = 'c:\\vystup.csv'

# hlavni pracovni trida
class main():

  # Funkce na osetreni neexistence klice slovniku
  def GetValue(self, slovnik, klic, vychozi):
    if klic in slovnik:
      return slovnik[klic]
    else:
      return vychozi

  # Vstupni bod
  def Run(self):

    FW = open(OUTFILE, 'w')

    dbv = VmDevPyVRDA.GetVRDADB()
    table = dbv.OpenRecordSet(u'F1')
    cursorPosition = table.CursorMove(CursorMoveType.First)

    while cursorPosition != CursorPosition.EOF:

      slovnikVemaData = table.RecordRead(True)

      oscis = str(self.GetValue(slovnikVemaData, u'oscis', u''))
      jmeno = self.GetValue(slovnikVemaData, u'jmeno', u'')
      prijmzd = self.GetValue(slovnikVemaData, u'prijmzd', u'')
      naroz = self.GetValue(slovnikVemaData, u'naroz', None).strftime("%d.%m.%Y")

      textRadekCSV = "%s;%s;%s;%s" % (oscis, jmeno, prijmzd, naroz)
      FW.write(textRadekCSV+"\n")

      print textRadekCSV

      cursorPosition = table.CursorMove(CursorMoveType.Next)

    table.Close()
    FW.close()


# Spusteni metody Run v instanci tridy main
if __name__ == "__main__":
  main().Run()
```

4.3.3 Skript pnaroz.py řádek po řádku

```
# -*- coding: utf-8 -*-
```

Skript bude psán v utf-8

```
import sys
import os
import VmDevPyVRDA
from VmDevPyVRDA import *
import PyVRDA
from PyVRDA import *
```

Příkazy import slouží k importu modulů externích knihoven s třídami a funkcemi. Samotný Python toho bez těchto modulů příliš neumí, jeho síla spočívá právě ve využívání toho, co někdo jiný už naprogramoval.

```
# Soubor pro vystup
OUTFILE = 'c:\\vystup.csv'
```

Zde se specifikuje cesta a soubor, do kterého se bude provádět výstup v CSV formátu. Každé opačné lomítko je nutné v Pythonu psát dvojitě, tak jak je uvedeno pro příklad c:\vystup.csv.

```
# hlavni pracovni trida
class main():
```

Aby bylo možno spouštět skripty pomocí DBV aplikace, je nutné všechnu požadovanou funkcionalitu obalit do běhové třídy. Zde není třeba cokoliv upravovat.

```
  # Funkce na osetreni neexistence klice slovniku
  def GetValue(self, slovnik, klic, vychozi):
    if klic in slovnik:
      return slovnik[klic]
    else:
      return vychozi
```

Jedná se o pomocnou metodu (funkci), která umí otestovat přítomnost konkrétního klíče v proměnné slovnik datového typu slovník. Pokud slovnik hledaný klíč obsahuje, funkce vrátí příslušnou hodnotu, pokud nikoliv, funkce vrátí obsah argumentu vychozi. Argument self souvisí s objektovým programováním a je nutné ho tam ponechat.

```
  # Vstupni bod
  def Run(self):
```

Jedná se o definici metody (funkce), která bude spuštěna jako první.

```
FW = open(OUTFILE, 'w')
```

Otevře se soubor c:\vystup.csv pro zápis. Hodnota „w" druhého argumentu říká, že bude smazát původní obsah souboru vystup.csv a pokud neexistuje, bude založen nový.

```
dbv = VmDevPyVRDA.GetVRDADB()
```

Do proměnné dbv se uloží reference na aktuální připojení k Vema databázi pro příslušnou aplikaci (PAM) Zde není třeba cokoliv upravovat.

```
table = dbv.OpenRecordSet(u'F1')
```

Otevření tabulky (souboru ve Vema databázi příslušné aplikace) F1. Do proměnné table se uloží příslušná reference. Otevření tabulky může obsahovat ještě několik dalších parametrů:

- table = dbv.OpenRecordSet(u'F1', filter=u'oscis<100 a zivi')
  - Jedná se zůžení obsahu tabulky (souboru) dle klasické Vema podmínky.
- table = dbv.OpenRecordSet(u'F1', writeAccess=True)
  - Povolení / zakázání zápisu nebo modifikace dat v otevřené tabulce (souboru)

```
cursorPosition = table.CursorMove(CursorMoveType.First)
```

Nastavení ukazatele záznamů v tabulce na záznam první.

```
while cursorPosition != CursorPosition.EOF:
```

Cyklus zajišťující procházení jednotlivých záznamů (řádků) otevřené tabulky.

```
    slovnikVemaData = table.RecordRead(True)
```

Načtení aktuálního záznamu (řádku, věty) do slovníku slovnikVema. Pro příklad je zde uveden první řádek tabulky F1, tak jak se v proměnné slovnikVemaData objeví:

{u'odkdy': datetime.datetime(2011, 10, 1, 0, 0), u'ospubl': 1L, u'jmeno': u'Jan', u'hzmen': datetime.datetime(2014, 1, 1, 0, 0), u'jmenozd': u'Jan', u'idcis': u'3257257511', u'prijmzd': u'Bondra', u'pohl': 0L, u'osc': 1L, u'korvyp': 28042L, u'zivmi': 3L, u'vicezam': 0L, u'statp': 203L, u'frm': 0L, u'tituly': u'Ing. CSc.', u'potpri': 1L, u'predvl': 1L, u'appvdr': 101L, u'prijm': u'Bondra Jan, Ing. CSc.', u'prezal': 0L, u'banv': 1L, u'form': 1L, u'redukce': 0L, u'oscis': 1L, u'pracv': 3L, u'prijmz': u'Bondra', u'upljmeno': u'Ing. Jan Bondra, CSc.', u'naroz': datetime.datetime(1952, 10, 30, 0, 0), u'cic': 2L, u'rocis': u'521030956', u'bancis': 1L, u'cicin': 0L, u'rozuc': 1L, u'titul': u'Ing. CSc.', u'cskup': 1L, u'indikace': 8L, u'zapl': datetime.datetime(2007, 12, 1, 0, 0)}

Výše uvedený slovník obsahuje všechny vyplněné položky daného řádku. Pokud je v souboru F1 nějaká položka nevyplněna (tj. NEDEF), ve slovníku se neobjeví ani její název

(klíč). Proto je při dalším zpracování nutné VŽDY ověřovat, zda-li je požadovaný klíč ve slovníku přítomný či nikoliv. Pokud o přečtení hodnoty neexistujícího klíče skončí s chybou.

```
oscis = str(self.GetValue(slovnikVemaData, u'oscis', u''))
```

Do proměnné oscis se uloží osobní číslo osoby, resp. hodnota položky s klíčem oscis ze slovníku slovnikVemaData. Ke zpracování je využito pomocné funkce GetValue, která otestuje přítomnost klíče oscis ve slovníku a vrátí buď osobní číslo nebo prázdný řetězec. Slovo self. souvisí s objektovým programováním a proto ze musí být.

```
jmeno = self.GetValue(slovnikVemaData, u'jmeno', u'')
prijmzd = self.GetValue(slovnikVemaData, u'prijmzd', u'')
```

Činnost je obdobná jako pro oscis.

```
naroz = self.GetValue(slovnikVemaData, u'naroz', None).strftime("%d.%m.%Y")
```

Činnost je obdobná jako pro oscis s tím rozdílem, že při nezadaném datu narození vrátí funkce GetValue None (obdoba NEDEF). Je to z důvodu následného formátování (na DD.MM.RRRR) datumu metodou strftime. Pokud by se datum nezformátovalo, do CSV souboru by se zapisoval ISO formát RRRR-MM-DD HH:MI:SS.

```
textRadekCSV = "%s;%s;%s;%s" % (oscis, jmeno, prijmzd, naroz)
```

Vytvoření řetězce, který se bude zapisovat do CSV souboru. Řetězec obsahuje 4 položky oddělené středníky. Zástupné znaky %s jsou nahrazeny hodnotami proměnných (oscis, jmeno, prijmzd, naroz) přesně v tomto pořadí.

```
FW.write(textRadekCSV+"\n")
print textRadekCSV
```

Zápis řetězce textRadekCSV do souboru společně se znakem nového řádku. Pro kontrolu i tisk na obrazovku.

```
cursorPosition = table.CursorMove(CursorMoveType.Next)
```

Posunutí ukazatele záznamu v tabulce F1 na další.

```
table.Close()
FW.close()
```

Uzavření přístupu k tabulce F1 a uzavření zápisu do souboru vystup.csv

```
# Spusteni metody Run v instanci tridy main
if __name__ == "__main__":
  main().Run()
```

Jedná se o servisní kód, který spouštěcímu mechanismu říká, že má spustit metodu Run() v objektu main.

Pokud nikde nenastane chyba, tak je výsledkem následující okno a soubor vystup.csv s tímtéž.