PROGRAMOVÉ STRUKTURY: ZÁKLADY PYTHONU

Vlastnosti, skripty, moduly, spuštění programu v Pythonu, operátory, sekvence, kolekce, jazykové konstrukce, funkcionální programování, uživatelské funkce, přehled modulů Pythonu

Python

- = Monty Python's Flying Circus
- česky Monty Pythonův létající cirkus

- Inspirováno seriálem ze 70 let 20 století
  - více na
    - http://www.imdb.com/title/tt0063929/?ref_=fn_al_tt_1
    - http://www.csfd.cz/film/70220-monty-pythonuv-letajici-cirkus/prehled/

Všeobecné vlastnosti

- Základní vlastností je snadnost použití
- Je stručný
- Je rozšiřitelný
  - časově náročné úseky lze vytvořit v C a vložit
- Lze přilinkovat interpret Pythonu k aplikaci napsané v C
- Má vysokoúrovňové datové typy
  - asociativní pole, seznamy
- Mnoho standardních modulů
  - pro práci se soubory, systémem, GUI, URL, regulární výrazy, …
- Strukturu programu určuje odsazování
- Proměnné se nedeklarují, má dynamické typy

Programy v pythonu

- Moduly
  - obsahuje pouze definice funkcí
  - lze srovnat s knihovnou
  - program `fibonacci.py`
    - obsahuje dvě funkce pro výpočet Fibonacciho čísel
- Skripty
  - obsahuje sekvenci příkazů, která je okamžitě vykonána (interpretována)
  - může obsahovat definice funkcí
  - program `citace.py`
    - vyžaduje textový soubor `jezek.txt`

Spuštění pythonu

- Z příkazové řádky – interaktivní mód
  - programem `python.exe`
  - nastavena cesta v proměnné `PATH`
  - naplněna systémová proměnná `PYTHONPATH` k nalezení programových modulů
- Příkazový mód
  - přímé spuštění python skriptu, např. `C:\skript.py`
- Vývojové prostředí IDLE (Python GUI)
- Plugin PyDev pro Eclipse

Spuštění – příkazová řádka


```
> python.exe
Python 3.5.1 …
Type "help", "copyright", "credits" or "license" for
more information.
>>>
>>> import fibonacci
>>> fibonacci.fib(5)
1
1
2
3
>>> exit()
>
```

Spuštění – IDLE

- Z nabídky Start spustíme `IDLE (Python 3.5)`
  - spustí běhové prostředí *Python Shell*
- Z nabídky shellu *File/Open* otevřeme `fibonacci.py`
- Otevře se okno editoru s tímto programem
- Vybráním nabídky editoru *Run/Run module* se program načte do *Python Shellu*
- Nyní můžeme v okně shellu zavolat funkci definovanou v modulu `fibonacci.py`

```
>>> fib(5)
```

Python jako kalkulačka

- Sčítání a odčítání

```
>>> 1 + 2   # součet celé číslo
3
>>> 1 + 2.0 # součet je reálné číslo
3.0
>>> 5 - 3   # rozdíl je celé číslo
2
>>> 5.0 - 3 # rozdíl je reálné číslo
2.0
```

- Násobení

```
>>> 3 * 6   # součin je celé číslo
18
>>> 3.0 * 6 # součin je reálné číslo
18.0
```

Python jako kalkulačka (2)

- Mocnina

```
>>> 3 ** 2  # druhá mocnina
9
>>> 2 ** 3  # třetí mocnina
8
```

- Dělení

```
>>> 15 / 2  # reálné dělení
7.5
>>> 15 // 2 # celočíselné dělení
7
>>> 15 % 2  # zbytek po celočíselném dělení
1
```

Python jako kalkulačka (3)

- Priorita operátorů a závorky

```
>>> 1 + 2 * 3    # priorita operátorů
7
>>> (1 + 2) * 3 # použití závorek
9
```

- Proměnné jako paměti
  - jsou case-sensitive

```
>>> vyska = 10
>>> sirka = 5.5 # výsledek bude reálné
                # číslo
>>> vyska * sirka
55.0
```

Python jako kalkulačka (4)

- Komplexní čísla

```
>>> c1 = 4 + 3j    # i je nahrazeno j
>>> c2 = (2 - 7j) # k. číslo v závorce
>>> c1 + c2        # součet
(6 - 4j)
>>> c1 - c2        # rozdíl
(2 + 10j)
>>> c1 * c2        # součin
(29 - 22j)
```

Velké celé číslo

- Python automaticky převádí celé číslo na něco, čemu se říká *velké celé číslo*
  - nezaměňujte s typem long v jazycích C a C++, kdy se číslo ukládá na 32 bitech
  - jde o specifickou vlastnost jazyka, která umožňuje pracovat s celými čísly o prakticky neomezené velikosti
  - platíme za to cenou mnohem pomalejšího zpracování

```
>>> 12345678912345678 9 * 123456789
1524157876543209975019052 1
```

Operátory

- Zkratkovité aritmetické operátory

```
>>> x = 3    # inicializace
>>> x += 6   # inkrementace
>>> x -= 4   # dekrementace
>>> x *= 2   # násobení
>>> x /= 3   # celočíselné dělení
>>> x %= 2   # zbytek po dělení
>>> x        # kolik bude x?
```

Operátory (2)

- Relační (porovnávací) operátory

```
>>> x = 2    # inicializace
>>> x < 5    # menší, odpoví True
>>> x > 5    # větší, odpoví False
>>> x == 5   # rovnost
>>> x != 5   # nerovnost
>>> x <= 5   # menší rovno
>>> x >= 5   # větší rovno
```

Operátory (3)

- Booleovské (logické) operace

```
>>> x = 3                       # inicializace
>>> x < 5 and x > 2             # logický součet
True
>>> (x < 5) and (x > 2)         # závorky nejsou
                                # třeba
True
>>> x > 5 or x < 2              # logický součin
False
>>> not x > 5                   # negace
True
```

Priorita operátorů

Operátor	Popis
`x ** y`	Mocnina
`-x, , +x, ~x`	Unární mínus, plus, jedničkový komplement
`*, /, //, %`	Násobení, (celočíselné) dělení, modulo
`+, -`	Sčítání, odčítání
`x << y, x >> y`	Posun o y bitů vlevo, vpravo
`x & y`	Bitový součin (and)
`x^y`	Bitová nonekvivalence (xor)
`x | y`	Bitový součet (or)
`<, <=, >, >=, ==, !=, in, not in, is, not is`	Porovnávání, testy členství a identity
`not x`	Logická negace
`x and y`	Logický součin
`x or y`	Logický součet

Sekvence

- Uspořádané množiny prvků
  - seznamy `['s', 'e', 'z', 'n', 'a', 'm']`
  - n-tice `('n', '-', 't', 'i', 'c', 'e')`
  - rozsahy
  - řetězce `"retezec"` nebo `'retezec'`
- Sekvence `S` sdílí společnou množinu operací
  - `len(S)` – velikost `S`, počet prvků `S`
  - `max(S)` – největší prvek `S`
  - `min(S)` – nejmenší prvek `S`

Sekvence (2)

- Další operace pro sekvenci S
  - `x in S` – test přítomnosti prvku `x` v `S`
  - `x not in S` – test nepřítomnosti prvku `x` v `S`
  - `S1 + S2` – konkatenace (zřetězení) sekvencí `S1` a `S2`
  - `S[i]` – `i`-tý prvek `S`, indexuje se od 0
  - `S[i:j]` – část sekvence začínající `i`-tým prvkem a končící před `j`-tým prvkem
  - `S[i:j:k]` – to samé, jen získám každý `k`-tý prvek
  - `S * n` – sekvence tvořena `n` kopiemi `S`
  - `tuple(S)` – převod členů `S` na n-tici
  - `list(S)` – převod členů `S` na seznam
  - `S.index(x)` – index prvního výskytu prvku `x` v `S`

Řetězce


```
>>> 'rovne ' + ('a vlevo ' * 3)  # operace + a *
'rovne a vlevo a vlevo a vlevo '
>>> s1 = 'ko '                   # řetězcová proměnná
>>> s2 = u'dak'                  # řetězec v Unicode
>>> s1 * 3 + s2                  # sestavení řetězce
'ko ko ko dak '
>>> str(20)    # převod libovolného objektu na řetězec
'20'
>>> 'kokodak'.find('ko')  # nalezení prvního výskytu
0
>>> 'kokodak'.replace('dak', 'kodak')  # náhrada
'kokokodak '
>>> 'co to tady je'.split()  # rozdělení řetězce do seznamu
['co', 'to', 'tady', 'je']
```

Řetězce (2)

- Řezy a výřezy řetězců

```
>>> s = 'testovaci retezec'
>>> s[0]   # první písmeno
>>> s[-1]  # poslední písmeno
>>> s[2:5] # odpověď je sto
>>> s[2:]  # od třetího znaku dál
>>> s[:5]  # do pátého znaku
```

- Řetězce na více řádek
  - uzavřen v trojitých uvozovkách (") nebo apostrofech (')

```
>>> s = """super
         dlouhy
         retezec"""
```

Chybné používání řetězců

- Silný typový systém

```
>>> 'cislo ' + 5
TypeError: Can't convert 'int' object
to str implicitly
```

  - nelze spojovat řetězec a celé číslo

- Změna prvku/znaku na dané pozici

```
>>> s = 'rada'
>>> s[0] = 'v'  # chci 'vada'
TypeError: 'str' object does not
support item assignment
```

Kolekce

- Nemusí mít prvky téhož typu

- Seznamy
- Množiny
- N-tice
- Slovníky
- Pole

Seznamy


```
>>> seznam = []  # vytvoří prázdný seznam
>>> jinySeznam = [1, 2, 3]
>>> jinySeznam
[1, 2, 3]
>>> jinySeznam[2] # prvek seznamu, od 0
3
>>> jinySeznam[2] = 7 # změna prvku
>>> jinySeznam
[1, 2, 7]
>>> jinySeznam[-2] # indexování od konce
2
```

Seznamy (2)


```
>>> seznam.append('neco')  # vložení prvku na konec
>>> seznam
['neco']
>>> seznam.append(jinySeznam) # vložit lze cokoliv
>>> seznam
['neco', [1, 2, 7]]
>>> seznam[1][2]     # lze idexovat uvnitř prvků
7
>>> del seznam[1]    # odstranění prvku
>>> seznam
['neco']
>>> seznam.append(5) # vložit lze i číslo
>>> seznam
['neco', 5]
```

Metody pro práci se seznamem

- Rozšíření seznamu o prvky jiného

```
>>> s1 = [1, 2 ,3]
>>> s2 = ['a', 'b', 'c']
>>> s1.extend(s2)
>>> s1
[1, 2, 3, 'a', 'b', 'c']
```

- Vložení prvku na danou pozici

```
>>> s2.insert(1, 'z')
>>> s2
['a', 'z', 'b', 'c']
```

Metody pro práci se seznamem (2)

- Odstranění prvku

```
>>> s2.remove('z')      # první výskyt prvku
>>> s2                  # výsledný seznam
['a', 'b', 'c']
>>> s1.pop(2)           # prvek na pozici
3                       # rušený prvek
>>> s1                  # výsledný seznam
[1, 2, 'a', 'b', 'c']
>>> s1.pop()            # poslední prvek
'c'                     # rušený prvek
>>> s1                  # výsledný seznam
[1, 2, 'a', 'b']
```

Metody pro práci se seznamem (3)

- Kopie seznamu

```
>>> s3 = s1.copy()  # s3 bude ['a', 'b', 'c']
```

- Smazání prvků seznamu

```
>>> s3.clear()      # s3 bude []
```

- Počet prvků `x` v seznamu `S`

```
>>> s = ['a', 'b', 'e', 'c', 'e', 'd', 'a']
>>> s.count('a')
2
```

- Obrácení prvků v seznamu

```
>>> s.reverse()
>>> s
['a', 'd', 'e', 'c', 'e', 'b', 'a']
```

Množina

- Nepatří mezi sekvence
  - nejsou uspořádané
  - založeny na seznamech
  - prázdná množina je prázdný seznam
- Množinové operace

```
>>> m = {}                  # prázdná množina
>>> n = {'a', 2, 3}         # definice množiny
>>> o = {1, 2, 3, 4, 4} # duplicitní prvky nevadí
>>> n.union(o)              # sjednocení
{1, 2, 3, 4, 'a'}
>>> n.intersection(o)    # průnik
{2, 3}
>>> n.difference(o)      # rozdíl
{'a'}
```

Metody pro práci s množinou

- Velikost množiny = počet prvků v množině

```
>>> m = {1, 2, 3, 4, 5}
>>> len(m)
5
```

- Test (ne)přítomnosti prvku v množině

```
>>> 2 in m
True
>>> 5 not in m
False
```

- Test podmnožiny a nadmnožiny

```
>>> m.issubset({1, 2, 3})    # podmnožina
True
>>> m.issuperset({1, 2, 3}) # nadmnožina
False
```

Metody pro práci s množinou (2)

- Vložení prvku do množiny

```
>>> m = {'p', 'g', 's'}
>>> m.add(2016)
>>> m
{2016, 'p', 's', 'g'}
```

- Zrušení prvku množiny

```
>>> m.remove('p')  # musí existovat
>>> m.discard('p') # není, neruší
>>> m.pop() # zruší libovolný prvek
2016
```

Metody pro práci s množinou (3)

- Kopie množiny

```
>>> n = m.copy()
>>> n
{'g', 's'}
```

- Smazání prvků množiny

```
>>> n.clear()
>>> n
{}
```

N-tice

- Posloupnost hodnot uzavřená do ()
  - lze s ní nakládat jako s celkem
  - nelze ji měnit
- Základní práce s n-ticemi

```
>>> ntice1 = (1, 2, 3, 4, 5)
>>> ntice1[1]
2
>>> ntice2 = ntice1 + (6,) # čárka navíc
                           # odlišuje číslo od n-tice
>>> ntice2
(1, 2, 3, 4, 5, 6)
>>> 4 in ntice2
>>> len(ntice2)
>>> max(ntice2)
```

Slovník

- Asociativní pole
  - realizováno hash tabulkami
  - klíče položek jsou neměnné
  - hodnota může být libovolného typu
- Vytvoření slovníku

```
>>> dict = {}
>>> dict['klic'] = 'hodnota'
>>> dict['klic']
'hodnota'
```

Slovník (2)

- Vytvoření slovníku (pokračování)

```
>>> adresy = { # naplněný slovník
'Mana' : ['Karlova 5', 'Plzen',
377123456],
'Blanka' : ['Za vsi', 'Praha', 222222222]
}
>>> adresy['Mana']
['Karlova 5', 'Plzen', 377123456]
>>> adresy['Blanka'][1] # hodnota je
    # seznam, kde lze indexovat
Praha
```

Metody pro práci se slovníkem

- Zjištění hodnoty prvku daného klíče

```
>>> adresy.get('Mana')
['Karlova 5', 'Plzen', 377123456]
```

- Test (ne)přítomnosti daného klíče

```
>>> 'Mana' in adresy     # True
>>> 'Mana' not in adresy # False
```

- Seznam všech hodnot

```
>>> adresy.values()
```

- Seznam všech klíčů

```
>>> adresy.keys()
```

- Velikost slovníku

```
>>> len(adresy)
2
```

Metody pro práci se slovníkem (2)

- Zrušení prvku ve slovníku

```
>>> del dict['klic']
```

- Kopie slovníku

```
>>> dct = seznamy.copy()
```

- Vymazání slovníku

```
>>> dct.clear()
```

- Převod na seznam s prvky [klíč, hodnota]

```
>>> adresy.items()
```

- Spojení slovníků

```
>>> adresy.update(dict)
```

- Iterátor nad klíči slovníku

```
>>> iter(adresy)
```

Zásobník

- Není explicitně definován
  - snadno realizován seznamem
    - metodou `append()` přidáme prvek na vrchol
    - metodou `pop()` odebereme prvek z vrcholu

```
>>> stack = []
>>> stack.append(1)
>>> stack.append('dva')
>>> stack.append(3)
>>> stack.pop()
3
>>> stack.pop()
'dva'
```

Fronta

- Není explicitně definována
  - snadno realizována seznamem
    - metodou `append()` přidáme prvek na konec fronty
    - metodou pop(0) vybereme prvek z čela fronty

```
>>> queue = []
>>> queue.append(1)
>>> queue.append('dva')
>>> queue.append(3)
>>> queue.pop(0)
1
>>> queue.pop(0)
'dva'
```

Pole

- Existuje modul `array`
- Prvky pole mohou být jen základních typů
  - znakové
  - číselné
- Používá se málo
- Nahrazuje se vestavěným seznamem

Tisk formátovaného řetězce

- Tisk vždy funkcí `print()`

```
>>> print('Hello World!')
```

- Formátování řetězce
  - sřetězováním částí do řetězcové proměnné

```
>>> pocet = 40
>>> veta = 'PGS navštěvuje ' + str(pocet) +
' studentů'
>>> print(veta)
```

  - formátováním uvnitř funkce `print()`
    - použitím operátoru % ve funkci `print()`
    - voláním funkce `format()` nad řetězcem

Formátování řetězce

- Starší způsob (verze 2.x)
  - parametry řetězce uvozeny %
    - znakové, řetězcové – `%c`, `%s`
    - číselné – `%d`, `%f`

```
>>> print('Hodnota %s je %f.' % ('pi', 3.14))
Hodnota pi je 3.140000
```

  - parametry definují
    - zarovnání – celé číslo za %
    - oříznutí – desetinná část za %

```
>>> print('Hodnota %s je %1.3f.' % ('pi', 3.14))
Hodnota pi je 3.140.
```

Formátování řetězce (2)

- Nový způsob (verze 3.x)
  - parametry jsou v { }
    - musí být číslovány od 0 nebo pojmenovány

```
>>> print('Hodnota {0} je {1}.'.format('pi', 3.14))
Hodnota pi je 3.14.
>>> print('Hodnota {konstanta} je
{hodnota}.'.format(konstanta = 'pi', hodnota = 3.14))
Hodnota pi je 3.14.
```

    - mohou definovat typ (jako u staršího způsobu)

```
>>> print('Hodnota {0:s} je {1:f}.'.format('pi', 3.14))
Hodnota pi je 3.140000.
```

    - mohou definovat zarovnání či oříznutí

```
>>> print('Hodnota {0:s} je {1:1.3f}.'.format('pi', 3.14))
Hodnota pi je 3.140.
```

Základní jazykové konstrukce

- Větvení programu
  - konstrukce `if-then-else`
  - konstrukce `if-then-elif-else`
- Cykly
  - cyklus `for`
  - cyklus `while`
- Prázdný příkaz
  - příkaz `pass`
- Odchycení a zpracování výjimek
  - konstrukce `try-except-finally`

Větvení programu


```
import random
cislo = random.randint(0, 100)

if cislo > 50:
    print('Velké číslo.')
else:
    print('Malé číslo.')
print('Mimo if, vytiskne se vždy.')
```

Větvení programu (2)


```
import random
cislo = random.randint(0, 100)

if cislo > 70:
    print('Velké číslo.')
else:
    if cislo > 30:
        print('Průměrné číslo.')
    else:
        print('Malé číslo.')
```

Větvení programu (3)


```
import random
cislo = random.randint(0, 100)

if cislo > 70:
    print('Velké číslo.')
elif cislo > 30:
    print('Průměrné číslo.')
else:
    print('Malé číslo.')
```

Cyklus for

- Cyklovat lze přes vše, co lze indexovat

```
for znak in 'cyklus':
      print(znak)

for prvek in [1, 'dva', 3]:
      print(prvek)

for slovo in ('jedna', 'dva', 'tri'):
      print(slovo)

for i in range(1, 20):
      print(i)
```

Cyklus for (2)

- Další definice rozsahu cyklu

```
mujSeznam = [1, 2, 3, 4]

for i in range(len(mujSeznam)):
      mujSeznam[i] += 1
      print(mujSeznam)

for index, hodnota in enumerate(mujSeznam):
      mujSeznam[i] += 1
      print(mujSeznam)

for x in mujSeznam[2:]:
      print(x)
```

Cyklus while

- Násobilka 7

```
j = 1
while j <= 10:
      print('7 * {0} = {1}.'.format(j, 7 * j))
      j = j + 1
```

- Malá násobilka

```
for i in range(1, 11):
      for j in range(1, 11):
            print('{0}*{1}={2}.'.format(i, j, i*j))
```

Další příkazy do cyklu

- Násilné ukončení cyklu
  - příkazem `break` v těle cyklu
  - ukončí cyklus, v němž je `break` voláno
- Vynucení nové iterace cyklu
  - příkazem `continue` v těle cyklu
- Provedení následné akce po ukončení cyklu
  - přidáním konstrukce `else`
  - provede se, pokud cyklus nebyl ukončen `break`

```
j = 1
while j <= 10:
    print('7 * {0} = {1}.'.format(j, 7 * j))
    j = j + 1
else:
        print('To byla násobilka 7.')
```

Odchyt a zpracování výjimek


```
# Hlídaný kus kódu
try:
      self.db = pyodbc.connect(self.dburl)

# Odchycení a obsluha definované výjimky
except pyodbc.Error as err:
      # nezadařilo se spojení s DB
      print("{0:s}".format(err))

# Vykoná se, až skončí úsek try
finally:
      sys.exit()      # ukončení programu
```

Funkcionální programování

- Definice uživatelské funkce
- Modul funkcí
  - zavedení modulu funkcí
  - zavedení vybraných funkcí z modulu
- Funkcionály
- Lambda výrazy
- Generátory seznamů

Uživatelská funkce

- Definice začíná klíčovým slovem `def`
  - může mít parametry
- Příkaz `return` slouží k předání návratové hodnoty
- Absence příkazu `return`
  - návratová hodnota je `None`
  - je to procedura
- Ukázkový příklad na výpočet *Fibonacciho* čísel
  - modul `fibonacci.py`

Modul funkcí

- Definice funkcí jsou sdružovány do modulů
- Moduly představují samostatné jmenné prostory
- Zavedení celého modulu
  - funkce nutno volat plně kvalifikovaným jménem

```
import fibonacci
fibonacci.fib(10)
```

- Zavedení vybraných funkcí modulu
  - funkce možno volat bez kvalifikace

```
from fibonacci import fib
fib(10)
```

- Při zavedení se vždy vykoná proveditelná část modulu

```
import citace
```

Funkcionály

- Funkce s argumentem typu funkce
- Mapovací funkce
  - `map(funkce, posloupnost)`
    - aplikuje funkci `funkce` na každý člen posloupnosti `posloupnost`
- Filtrační funkce
  - `filter(funkce, posloupnost)`
    - vybírá všechny prvky posloupnosti `posloupnost`, pro které funkce `funkce` vrací hodnotu `True`

Funkce map()


```
>>> def cube(x):
    return x ** 3

>>> strany = [1, 2, 3, 4]
>>> objemy = map(cube, strany)
>>> list(objemy)
[1, 8, 27, 64]
```

Funkce filter()


```
>>> def lichost(x):
    return (x % 2 == 1)

>>> cisla = [1, 8, 27, 64]
>>> licha = filter(lichost, cisla)
>>> list(licha)
[1, 27]
```

Lambda výrazy

- Definují anonymní (bezejmenné) funkce
- Mají tvar:

```
lambda <seznam parametrů>: <výraz>
```

```
>>> s = [1, 2, 3, 4, 5]
>>> list(map(lambda x, y, z: x + y + z, s, s, s))
[3, 6, 9, 12, 15]
>>> mojefce1 = lambda x, y: x * y
>>> mojefce2 = lambda x: x + 1
>>> mojefce1(mojefce2(5), 6)
36
```

Generátory seznamů

- Pro vytváření nových seznamů
  - konstrukce `if` je nepoviná

```
seznam = <výraz> for <proměnná> in
<kolekce> if <podmínka>
```

- Můžeme vyjádřit také

```
seznam = []
for proměnná in kolekce:
    if podmínka:
        seznam.append(výraz)
```

Generátory seznamů (2)

- Seznam sudých čísel do 10

```
>>> [n for n in range(10) if n % 2 == 0]
[0, 2, 4, 6, 8]
```

- Kvadratická čísla do základu 5

```
>>> [n * n for n in range(5)]
[0, 1, 4, 9, 16]
```

- Výběr prvků z jiného seznamu

```
>>> hodnoty = [1, 13, 25, 7]
>>> [x for x in hodnoty if x < 10]
[1, 7]
```

Přehled modulů Pythonu

- Seznam proměnných a názvů funkcí zavedených modulů do paměti

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

- Seznam proměnných a názvů funkcí daného objektu

```
>>> objekt = 'Python'
>>> dir(objekt)
['__add__', '__class__', '__contains__',
'__delattr__', '__dir__',
…,
'title', 'translate', 'upper', 'zfill']
```

Přehled modulů Pythonu (2)

- Python obsahuje moduly pro práci s
  - řetězci
  - databázemi
  - datumy
  - numerikou
  - internetem
  - značkovacími jazyky
  - formáty souborů
  - kryptováním
  - adresáři a soubory
  - komprimováním
  - perzistencí

Přehled modulů Pythonu (3)

- Python obsahuje moduly pro práci s (pokrač.)
  - generickými službami
  - službami operačního systému
  - meziprocesovou komunikací síťováním
  - internetovými protokoly
  - multimediálními službami
  - GUI
  - multijazykovými prostředky
  - a další

Modul sys

- Specifické parametry a funkce Pythonu
- `exit()` – ukončen běhu programu
- `argv` – seznam argumentů z příkazového řádku
- `path` – seznam cest prohledávaných při práci s moduly
- `platform` – identifikuje platformu
- `modules` – slovník již zavedených modulů
- a další

Modul os

- Interakce s operačním systémem
- `name` – kód označující jádro operačního systému
- `system()` – volání příkazu operačního systému
- `mkdir()` – vytvoření adresáře
- `getcwd()` – zjištění aktuálního pracovního adresáře

```
>>> chdir('D:\pgs')
>>> getcwd()
'D:\\pgs'
```

- a další

Modul re

- Práce s regulárními výrazy
- `search()` – hledej vzorek kdekoliv v řetězci
- `match()` – hledej pouze od začátku řetězce
- `finall()` – najdi všechny výskyty
- `split()` – rozděl na podřetězce, které jsou odděleny zadaným vzorkem
- `sub()`, `subn()` – náhrada řetězců
- a další

Modul math

- Poskytuje matematické funkce a konstanty
- `sin()`, `cos()` – goniometrické funkce
- `log()`, `log10()` – přirozený a dekadický logaritmus
- `ceil()`, `floor()` – zaokrouhlování nahoru a dolů
- `sqrt()` – druhá odmocnina
- `pi`, `e` – známé konstanty
- a další

Modul time

- Práce s datem a časem
- `time()` – vrací aktuální čas v milisekundách
- `gmtime()` – převod času v milisekundách na UTC (GMT)
- `localtime()` – převod do lokálního času vůči UTC
- `mktime()` – opak `localtime`
- `sleep()` – zastaví běh programu na zadaný počet sekund
- a další

Modul random

- Generátor náhodných čísel
- `seed()` – počáteční nastavení generátoru
- `random()` – náhodné reálné číslo z intervalu 0..1
- `randint()` – náhodné celé číslo z intervalu (včetně)
- `sample()` – generování náhodného podseznamu z daného seznamu
- a další

Práce se soubory

- Otevření souboru
  - `f = open(soubor, [mod])`
  - časté módy otevření souboru
    - r, rt, rb – pouze pro čtení
    - w, wt, wb – pouze pro zápis, předchozí obsah smaže
    - r+, r+b – aktualizace souboru
    - a – zápis na konec (existujícího) souboru
- Přečtení celého souboru
  - `retezec = f.read()`
  - uložen do řetězce
- Přečtení maximálně dalších n znaků/bytů
  - `f.read(n)`

Práce se soubory (2)

- Přečtení řádky za souboru
  - `f.readline()`
    - načte a vrátí jeden řádek zakončený \n jako řetězec
  - `f.readlines()`
    - načte a vrátí seznam načtených řádek do konce souboru
- Zápis řetězce `s` do souboru `f`
  - `f.write(s)`
- Zápis seznamu řetězců `L` do souboru `f`
  - `f.writelines(L)`
- Uzavře soubor `f`
  - `f.close`

Program pocet1.py


```
def pocetSlov(s):
    seznam = s.split()  # Rozdělí řetězec na jednotlivá slova
    return len(seznam)  # Vrátíme počet prvků (slov) seznamu

vstup = open("jezek.txt", "r") # Otevře soubor pro čtení
celkem = 0  # Vytvoříme a vynulujeme proměnnou

# V cyklu zpracuj každý načtený řádek souboru
for radek in vstup.readlines():
    # Sečti počty slov za každý řádek
    celkem = celkem + pocetSlov(radek)

print('Soubor má {0} slov.'.format(celkem))
vstup.close()
```

Program pocet2.py


```
import sys
if len(sys.argv) != 2:   # Zadání názvu souboru z klávesnice
    soubor = input("Zadejte název souboru: ")
else:
    soubor = sys.argv[1] # Soubor předán jako parametr skriptu

vstup = open(soubor, "r")
radku = slov = znaku = 0

for radek in vstup.readlines():
    radku = radku + 1             # Počet řádků souboru
    seznamSlov = radek.split()
    slov = slov + len(seznamSlov) # Sčítání počtů slov každého řádku
    znaku = znaku + len(radek)    # Sčítání počtu znaků každého slova

print("{0:s} má {1:d} řádků, {2:d} slov a {3:d} znaků.".format(soubor, radku, slov,
znaku))
vstup.close()
```