Skriptovací jazyky

Použití tradičních jazyků

- Pro vytváření programů „z gruntu“
- Obvyklé je použití v týmu
- Velké, řádně specifikované a výkonnostně vyspělé aplikace

} mělo by to tak být, ale jako vždy skutečnost je trochu jiná

Použití skriptovacích jazyků

- K vytváření aplikací z předpřipravených komponent
- K řízení aplikací, které mají programovatelný interface
- Ke psaní programů, je-li rychlost vývoje důležitější než efektivita výpočtu
- Jsou nástrojem i pro neprofesionální programátory

Vznik 70 léta Unix „shell script“´= sekvence příkazů čtených ze souboru a prováděných jako by z klávesnice ostatní to převzali (AVK script, Perl script, …)

Skript = textový soubor určený k přímému provedení (k interpretaci)

Vlastnosti

•Integrovaný překlad a výpočet
•Nízká režie a snadné použití (např. impl.deklarace)
•Zduřelá fukčnost ve specif. oblastech (např. řetězce a regul. výrazy)
•Není důležitá efektivita provedení (často se spustí je 1x)
•Nepřítomnost sekvence překlad-sestavení-zavedení

Tradiční použití

•Administrace systémů (sekvence shell příkazů ze souboru)
•Vzdálené řízení aplikací (dávkové jazyky)

Nové použití

•Visual scripting = konstrukce grafického interface z kolekce vizuálních objektů (buttons, text, canvas, back. a foregr. colours, …). (Visual Basic, Perl)
•Lepení vizuálních komponent (např spreadsheet do textového procesoru)
•Dynamické Web stránky = dynamické řízení vzhledu Web stránky a jednoduchá interakce s uživatelem, která je interpretována prohlížečem
•Dynamicky generované HTML = část nebo celá HTML je generována skriptem na serveru. Používá se ke konstrukci stránek, jejichž obsah se vyhledává z databáze.

Prostředky Web skriptů: VBScript, JavaScript, JScript, Python, Perl, …

Python - vlastnosti

Název Monty Python’s Flying Circus

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), dynamické typování

Mnoho standardních modulů (pro práci se soubory, systémemí, GUI, URL, reg.výrazy, …)

Strukturu programu určuje odsazování

Proměnné se nedeklarují, má dynamické typy

Python - spouštění

Spuštění

- **V příkazovém okně zápisem *python* (musí být cesta na python.exe a pro nalezení programových modulu naplněna systémová proměnná PYTHONPATH. Implicitne vidi do direktory kde je spuštěn). Např.**

cmd → cd na direktory se soubory .py → c:\sw\python25\python.exe → import P1fibo → P1fibo.fib(5) Ukončení Pythonu exit() nebo Ctrl-Z a odřádkovat

interaktivní zadávání příkazů lze provádět ihned za promt >>> tzv.interaktivní mód

- **Nebo příkazový mód c:\sw\python25>skript.py**
- **Alternativou je použití oken IDLE (Python GUI) po jeho spuštění se objeví oknoPythonShell ve tvaru:**

```
Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
    ****************************************************************
        Nějaké anglické řeči
    ****************************************************************
 IDLE 1.2
>>>
```

V jeho záhlaví vybereme File → NewWindow → vznikne okno Untitlet

V záhlaví Untitled vybereme File→Open a otevřeme např P1fibo.py soubor, pak Run→RunModule

>>>fib(5)

V oknech lze editovat, před spuštěním ale ulož.

Všimněte si rozdílu spuštění modulu P1fibo - definuje funkce, které musím vyvolat

P2cita – je přímospustitelný kód >>>import P2cita.py

Python - čísla

Interaktivní použití připomíná komfortní kalkulačku s proměnnými, typy a funkcemi. Napr.

```
>>>print "Monty Python's " + " Flying Circus"   # pro řetězce lze použít “ i  ’ ale v párech
Monty Python's Flying Circus
>>> print 'Ahoj' + 1    #  má silný typový systém
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
>>> print 15/2          #celočíselné
7
>>> print 15/2.0        #reálné,              pro získání zbytku po dělení je operátor %
7.5
>>> vyska = sirka = 10.5 ** 2   #násobné přiřazení, umocňování
>>> plocha = vyska*sirka        #jména jsou case sensitive
>>> plocha*(5.5+12)
212713.59375
>>> print "Soucet cisla %d a cisla %d je: %d" % (vyska, sirka, vyska+sirka) #formátování
      najdete v textu
Soucet cisla 110 a cisla 110 je: 220
>>> Imag = (1 - 1J)*-1j                    # umí komplexní čísla
>>> Imag
(-1-1j)
```

Python - operátory

Python automaticky převádí celé číslo na něco, čemu se říká *velké celé číslo* (*long integer*; 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
15241578765432099750190521L
```

```
>>> x += 1 #je použitelný zkratkový zápis výrazů M += N, M -= N, M *= N, M /= N, M %= N
```

Relační operátory ==, >, <, != nebo <>, <=, >=

Booleovské hodnoty True, False

Booleovské operace and, or, not

Artmetické operátory zahrnují mocnění

```
>>> 2**2**3          pozor, je pravoasociativní
256
>>> (2**2)**3
64
```

Python - sekvence

Sekvence jsou uspořádané množiny prvků

Zahrnují 8bitové řetězce, unikódové řetězce, seznamy a n-tice

Všechny sekvence S sdílí společnou množinu funkcí:

- len(S) počet prvků
- max(S) největší prvek
- min(S) nejmenší prvek
- x in S test přítomnosti x v S
- x not in S test nepřítomnosti
- S1 + S2 konkatenace dvou sekvencí
- tuple(S) konverze jakékoliv sekvence S na n-tici
- list(S) konverze S na seznam
- S*n nová sekvence tvořená n kopiemi S
- S[i] i-tý prvek S, počítá se od 0
- S[i:j] část sekvence S začínající i-tým prvkem až do j-tého bez něj

Pár př.

t = tuple(‘100’) $\Rightarrow$ t = (‘1’, ‘0’, ‘0’)

l = list(‘100’) $\Rightarrow$ l = [‘1’, ‘0’, ‘0’]

Pozn. Příklady lze do pythonu překopírovat i rovnou z PowerPointu

Python - řetězce

Řetězce

```
>>> print 'vlevo '+ ('a vlevo' * 3)          #operátor  '+' zřetězuje  a ' * ' opakuje
 vlevo a vlevo a vlevo a vlevo
>>> s1 = 'ko '                               #má řetězcové proměnné
>>> s2 = u'dak '                              # u je to v Unicode
>>> print s1*3 + s2
ko ko ko dak
>>>s1=str(20)                    # pro převod jakéhokoliv objektu na řetězec
>>> s1
‘20’
>>> unicode('\x51\x52\x53')     # pro převod na Unicode řetězce hexad. znaků
u'QRS'
>>> s = ‘CSc’
>>>print s.find(‘Sc')                         # vyhledání v textu
1                                              najde začátek podřetězce
>>>
print s.replace(‘CSc', ‘PhD')      # náhrada textu
‘PhD’
>>>'co to tady je'.split()          # split vrací seznam řetězců, může mít parametr=vypouštěný řetěz
['co', 'to', 'tady', 'je']
```

A mnoho dalších funkcí

Python - řetězce

```
>>> s1 = 'to je moje chyba‘       #má řezy a výřezy řetězců, čísluje od 0
>>> s1[1:3]
'o ‘                        je to o a mezera
>>> s1[3:]                  # od třetího dál
'je moje chyba'
>>> s1[:3]
'to ‘                       t, o a mezera
>>> s1[3]
'j'
>>> s1[0] = ‘6’                     #je chyba, 'str' object does not support item assignment
>>> len(s1)                         # vrací délku řetězce
16

#Řetězce přes více řádek píšeme mezi “““     “““
>>>print"""sssssssssssssssssssssssssjjjjjjjjjjjjjjjjjjjjjjj
jjjjjjjjjjjjjjjjjjjjjjj"""
sssssssssssssssssssssssssjjjjjjjjjjjjjjjjjjjjjjj
jjjjjjjjjjjjjjjjjjjjjj
>>>
```

Python – kolekce – seznamy

kolekce v Pythonu nemusí mít prvky téhož typu, zahrnují seznamy, n-tice, slovníky, pole, množiny

Konstruktorem seznamů jsou [ ]

```
>>> seznam = []
>>> jinySeznam = [1, 2, 3]
>>> print jinySeznam
[1, 2, 3]
>>> print jinySeznam[2]          # čísluje od 0
3
>>> jinySeznam[2] = 7            # oproti řetězci je seznam modifikovatelný (mutable)
>>> print jinySeznam             # což znamená, že může měnit své prvky
[1, 2, 7]
 >>> print jinySeznam[-1]        # Záporné indexy indexují od konce a  stejně tak je to i u řetězců
7


>>> r='12345'
>>> r[-2]
'4'
>>>
```

Python – kolekce - seznamy

append je operátor (metoda), která připojuje svůj argument na konec seznamu

```
>>> seznam.append(“neco”)          #pokračujeme s hodnotami z předchozího slidu
>>> print seznam
[‘neco’]
>>> seznam.append(jinySeznam) #appendovat lze libovolny objekt
>>> print seznam
[‘neco’, [1, 2, 7]]
>>> print seznam[1][2] #v seznamech lze indexovat
7
>>> adresy = [['Mana', 'Karlov 5', 'Plzen', '377231456'],
        ['Pepina', 'V ZOO 42', 'Praha', '222222222']]
>>> adresy[1][2]
‘Praha’
>>> del seznam[1]  #odstraneni indexem zadaneho prvku
>>> print seznam
[‘neco’]
>>> seznam.append(“neco”)
print seznam
[‘neco’, 5]
```

Python – kolekce - seznamy

Spojení seznamů provedeme operátorem +

```
>>> novySeznam = seznam + jinySeznam
>>> print novySeznam
[‘neco’, 1, 2, 7]
>>> print [1,3,5,7].index(5)      #lze zjistit index prvku
 2
>>> len(adresy)      #lze zjistit delku seznamu
2
```

Další metody pro práci se seznamy

extend(L) přidá na konec prvky seznamu L ; dtto s.[len(s):] = L
insert(i, x) vloží prvek i na pozici x
remove(x) odstraní první výskyt prvku x v seznamu
pop(i) odstraní prvek na pozici i a vrátí jeho hodnotu
pop() odstraní poslední prvek a vrátí jeho hodnotu
count(x) vrátí počet prvků s hodnotou x
sort() seřadí prvky dle velikosti (modifikuje seznam), výsledek nevrací
reverse() obrátí ho, nevrací výsledek

```
>>> L=['co', 'to', 'tady', 'je']
>>> L.reverse()
>>> L
['je', 'tady', 'to', 'co']
```

Python – kolekce - seznamy

```
>>> s = [1,2,3]
>>> s
[1, 2, 3]
>>> s[len(s):] = [9,8,7]   # části s od indexu: včetně nahradí zadaným seznamem
>>> s
[1, 2, 3, 9, 8, 7]
>>> s[1:3]  # lze dělat řezy a výřezy v seznamech jako v řetězcích
[2,3]
>>> s = [6,3.4,'a']
>>> s.sort()
>>> s
[3.3999999999999999, 6, 'a']
>>> s = [1,2,3,4,5]
>>> s.pop(1)
2
>>> s.pop(-1)
5
>>> s.pop()
4
>>> s=[1.1,2.2,3.3,4.4,5.5]
>>> del s[2]
>>> s
[1.1000000000000001, 2.2000000000000002, 4.4000000000000004, 5.5]
```

Python – kolekce - množina

Množiny nepatří mezi sekvence protože nejsou uspořádané. Jsou ale založeny na seznamech, prázdná množina je prázdný seznam

```
>>> import sets       # Set je v modulu sets, od verze 2.5 je součástí jazyka jako set
>>> M=sets.Set()
>>> N=sets.Set(['a',2,3])          # Set provede převedení seznamu na množinu
>>> O=sets.Set([1,2,3,4])
>>> U= N.union(O)
>>> U
Set(['a', 1, 2, 3, 4])
>>> U.intersection(N)
Set(['a', 2, 3])
>>> U.intersection(N) == N
True
>>> sets.Set([2,3]).issubset(N)              # test je-li podmnožinou N.   [2,3] se musí konvertovat
True
>>> U.issuperset(N)
True
>>> 2 in O
True
```

Python – kolekce – N-tice

N-tice Pythonu je posloupnost hodnot uzavřená do ( ), lze s ní nakládat jako s celkem.

Po vytvoření nelze n-tici měnit (oproti seznamům jsou „immutable“)

Na jejich prvky lze odkazovat indexem

```
>>> ntice = (1, 2, 3, 4, 5)
>>> print ntice[1]
 2
>>> ntice1 = (1, 2, 3)
>>> ntice2 = ntice1 + (4,)      # čárka způsobí, že se zápis chápe jako n-tice a ne jako číslo
>>> print ntice2
(1, 2, 3, 4)
>>> 2 in (1,2,3)
True
>>> len((1,2,3))
3
>>> max((1,2,3,4,5,(2,2)))
(2, 2)
>>> min((1,2,3,4,5))
1
```

Python – kolekce – slovníky (dictionary)

Jsou to asociativní pole. Položky jsou zpřístupněny pomocí klíčů immutable typu, hodnotou může být libov. typ

Realizují se hash tabulkami. Konstruktorem jsou { }.

Konstruktorem je také dict([dvojice,dvojice, …])

```
>>> dct = {}            # na pocatku třeba mame prazdnou tabulku
>>> dct['bubu'] = ‘klicem je retezec a hodnota je take retezec’ # do ni muzeme vkladat
>>> dct['integer'] ='Celé číslo‘
>>> print dct['bubu']
klicem je retezec a hodnota je take retezec
>>> dct[3]=44.5
>>> print dct[3]
44.5
>>>adresy = {          # naplnme oba sloupce slovníku
            'Mana' : ['Karlova 5', 'Plzen', 377123456],
            'Blanka' : ['Za vsi', 'Praha', 222222222] }
>>> print adresy ['Mana']
['Karlova 5', 'Plzen', 377123456]
>>> mesto = 1
>>> print adresy['Blanka'] [mesto]          # můžeme indexovat v části hodnota (ta je seznamem)
Praha
>>>
```

Python – kolekce – slovníky (dictionary)

```
>>>adresy.get('Mana')            # zjistí hodnotovou část
['Karlova 5', 'Plzen', 377123456]
>>> adresy.has_key('Blanka')
True
>>> adresy.values()
[['Karlova 5', 'Plzen', 377123456], ['Za vsi', 'Praha', 222222222]]
>>> dct.update(adresy)          #spoji slovniky dct a adresy
>>>a =dct.copy()                #nakopiruje dct do a, stačí napsat a = dct
>>>len(a)
4
>>> del adresy['Mana']          #odstraneni polozky s danym klicem
>>> print adresy.keys()         #tiskne seznam vsech klicu
[‘Blanka']
>>> adresy.clear()              #vyprazdni slovnik adresy
>>> adresy.items ()             #navraci seznam vsech polozek slovniku adresy
[]
>>> dct.items()
[('bubu', 'klicem je retezec a hodnota je take retezec'), ('integer', 'Cel\xe9 \xe8\xedslo'), (3, 44.5), ('Blanka', ['Za
    vsi', 'Praha', 222222222]), ('Mana', ['Karlova 5', 'Plzen', 377123456])]
```

Python – kolekce – slovníky (dictionary)

Operace na slovnících

- len(D) vrací počet položek slovníku D
- D[k] vrací hodnotu s klíčem k, neexistuje-li vyvolá exception
- D[k] = v dosadí do položky s klíčem k hodnotu v
- del D[k] odstraní položku s klíčem k, neexistuje-li vyvolá exception
- D.has_key(k) true, když tam klíč k je
- D.items() seznam (klíč, hodnota) z D
- D.keys() seznam všech klíčů z D
- D.values() seznam všech hodnot z D v pořadí jako D.keys()
- D.update(E) spojí slovníky E a D, při stejných klíčích použije hodnoty z E
- D.get(k, x) vrací hodnotu D[k] , neexistuje-li, vrací x nebo None při neuvedení
- D.setdefault(k [, x]) ----------“----------“----------, a dosadí ho do D[k]
- D.iteritems() vrací iterátor nad dvojicemi (klíč, hodnota) D
- D.iterkeys() -------“---------- klíči D
- D.itervalues -------“---------- hodnotami D
- D.popitem() vrátí a odstraní z D libovolnou položku. Při prázdném D vyvolá exception
- x in D True je-li x klíčem v D
- x not in D obráceně

Python – kolekce – slovníky (dictionary)

Př.

```
>>> D= {1:'jedna', 2:'dva', 3:'tri', 4:'ctyri'}
>>> D.popitem()
(1, 'jedna')
>>> for l in D.itervalues(): print l

dva
tri
Ctyri
>>> for l in D.iterkeys(): print l

2
3
4
>>>for l in D.iteritems(): print l

(2, 'dva')
(3, 'tri')
(4, 'ctyri')
```

Python – zásobník

Není tam explicitně typ zásobník, ale snadno se realizuje seznamem

Pomocí append() přidáme na vrchol

a pomocí pop() odebereme z vrcholu

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

Python - fronta

Explicitně není, ale snadno se realizuje zásobníkem
Pomocí append() přidáme na konec fronty
a pomocí pop(0) odebereme z čela fronty

```
>>> queue = ['prvni', 'druhy', 'treti']
>>> queue.append("posledni")
>>> vylezlo = queue.pop(0)
>>> print vylezlo
prvni
>>>
```

Python - pole

Existuje modul **array.**

Jeho prvky mohou být jen základních typů (znakové, integer, real)

Používá se málo, nahrazuje se vestavěným typem **seznam.**

Python -cykly

for in konstrukcí lze cyklovat přes vše co lze indexovat. Pozor na mezery

```
>>> for i in range(1, 20):
        print "%d x 120 = %d" % (i, i*120)          #co to tiskne? 20 radku tvaru 1 x 10 = 120
>>> for znak in ' retezec ': print znak
>>> for slovo in ('jedna', ‘dva', 'dva', ‘tri', ‘nic'): print slovo
>>> for prvek in ['jedna', 2, 'tri']: print prvek
```

Nechť seznam obsahuje odkazy na objekty 1,2,3,4, při jeho změně tam lze dát jiná čísla

```
mujSeznam = [1, 2, 3, 4]
for index in range(len(mujSeznam)):
   mujSeznam[index] += 1
   print mujSeznam              #tiskne [2, 2, 3, 4] pak [2, 3, 3, 4], pak [2, 3, 4, 4] a [2, 3, 4, 5],
```

Enumerate zajistí, že při každém průběhu máme dvojici index, hodnota

```
mujSeznam = [9, 7, 5, 3]
for index, hodnota in enumerate(mujSeznam):
     mujSeznam[index] = hodnota + 1
     print mujSeznam                                #tiskne stejny efekt jako predchozi
```

Řezy a výřezy jsou také dovoleny

```
for x in mujSeznam[2:] : print x                    #tiskne 6 a pak 4
```

Python - cykly

```
j = 1
while j <= 12:                          odděluje řídící řetězec od výrazů
    print "%d x 12 = %d" % (j, j*12)          # d=c.cislo, f=desetinne, e=s exponentem, s=retezec
    j = j + 1
else: print 'konec'     # else cast je nepovinna
```

Při vnořování pozor na správné odsazování

```
for nasobitel in range(2, 13):
     for j in range(1, 13):
              print "%d x %d = %d" % (j, nasobitel, j*nasobitel)
```

break jako v C přeruší nejblíže obepínající cyklus

continue pokračuje sdalší iterací cyklu

Cyklus while i for může mít else část, která se provede, jeli cyklus skončen (nesmí ale skončit break)

větvení

```
if j > 10:
     print "Toto se mozna vytiskne"
elif j == 10:                    #těch elif částí může být libovilně
     print "Toto se pak nevytiskne"
else:                             #else část je nepovinná
     print "Toto se mozna nevytiskne"

seznam = [1, 2, 3, 0, 4, 5, 0]          #program nic.py
index = 0
while index < len(seznam):
    if seznam[index] == 0:
        del seznam[index]
    else:
        index += 1
print seznam
```

Prázdný příkaz

```
pass
```

Python – precedence operátorů

x **y	mocnina
-x, ~x	minus, jedničkový komplement
*, /, %	krát, děleno, modulo
+, -	
x<<y, x>>y	posun o y bitů vlevo, vpravo
x&y	and po bitech
x^y	exklusivni or po bitech
x|y	or po bitech
<, <=, ==, >=, !=, x in y, x not in y, x is y, x is not y	porovnání , testy členství a identity
not x	booleovské not
x and y	booleovské and
x or y	booleovské or

Python – funkcionální programování

Definice funkcí

```
# Fibonacciho čísla – modul    - soubor P1fibo.py
def fib(n):    # vypise Fibonacciho cisla do n
    a, b = 0, 1  #vícenásobné přiřazení, počty musí být stejné
        while b < n:
        print b,
        a, b = b, a+b

def fib2(n): # vypise Fibonacciho cisla do n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result

>>> def nic():
      pass  # to je prazdny prikaz

>>> print nic()  #Funkce, které nemají return mají hodnotu None
None
```

definuje fci

definuje fci

Python – funkcionální programování

Definice fcí jsou sdružovány do modulů (modules)

Moduly představují samostatné prostory jmen.

Interpret příkazem import shellu nebo spuštěním v IDLE se modul zavede

```
>>> import P1fibo                # objekty z modulu se ale musi psat    P1fibo.fib(3)
```

nebo

```
>>> from P1fibo import fib       # objekt fib se jiz nemusi kvalifikovat
```

nebo

```
>>> from P1fibo import *         # objekty z modulu se nemusi kvalifikovat
```

nebo

```
>>> from P1fibo import fib2, fib
>>> fib(9)
1 1 2 3 5 8
```

Moduly mohou obsahovat libovolný kód, nejen definice fcí

```
>>> import P2cita
```

Všechny proveditelné příkazy se při zavlečení příkazem import ihned provedou

Python – funkcionální programování

Python obsahuje i funkcionály (funkce s argumentem typu funkce, více v LISPu)

map(funkce, posloupnost) Tento funkcionál aplikuje pythonovskou funkci **funkce** na každý z členů posloupnosti **posloupnost** (t.j seznam, n-tice, řetězec) . Př.

```
>>> list = [1,2,3,4,5]              # lze overit kopirovanim po radcich do IIDLE
>>> def cube (x):
    return x*x*x


>>> L = map(cube, list)
>>> print L
[1, 8, 27, 64, 125]
```

filter(funkce, posloupnost) vybírá všechny prvky posloupnosti **posloupnost**, pro které **funkce** vrací hodnotu True.

```
>>> def lichost(x): return (x%2 !=0)

>>> print filter(lichost, L)
[1, 27, 125]
```

Python – funkcionální programování

reduce(funkce, posloupnost) redukuje **posloupnost** na jedinou hodnotu tím, že na její prvky aplikuje danou binární **funkci**.

```
>>> def secti(x,y):return x+y


>>> print reduce (secti,L)
225
>>> L
[1, 8, 27, 64, 125]
```

Python – funkcionální programování

Lambda výrazy definují anonymní fce (tj. bez jména, převzato z LISPu) zápisem:

lambda <seznam parametrů> : < výraz >

Př.

```
>>> L=[1,2,3,4,5,6]
>>> print map(lambda x,y,z: x+y+z, L,L,L)
[3, 6, 9, 12, 15, 18]
>>> mojefce1 = lambda x,y:  y*x
>>> mojefce2 = lambda x: x+1
>>> print mojefce1(mojefce2(5),6)
36
```

Generátory seznamů - pro vytváření nových seznamů

[<výraz> for <proměnná> in <kolekce> if <podmínka>]

Můžeme vyjádřit ekvivalentním zápisem:

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

Např.

```
>>> [n for n in range(10) if n % 2 == 0 ]  #generuje sudá čísla do 10. % je operator modulo
[0, 2, 4, 6, 8]
>>> [n * n for n in range(5)]     #if část je nepovinná,
[0, 1, 4, 9, 16]
>>> hodnoty = [1, 13, 25, 7]
>>> [x for x in hodnoty if x < 10]   #v casti kolekce muze byt samozrejme i explicitni seznam
[1, 7]
```

Výpis jmen proměnných a fcí platných v daném modulu provede fce dir(jménomodulu)
„ všech „ „ provedeme fcí dir()

Python obsahuje moduly pro práci s:

- Řetězci
- Databázemi
- Datumy
- Numerikou
- Internetem
- Značkovacími jazyky
- Formáty souborů
- Kryptováním
- Adresářemi a soubory
- Komprimováním
- Persistencí
- Generickými službami
- OS službami
- Meziprocesovou komunikací a síťováním
- Internetovými protokoly
- Multimediálními službami
- GUI
- Multijazykovými prostředky
- A další

sysUmožňuje interakci se systémem Python:

- exit() — ukončení 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

os Umožňuje interakci s operačním systémem:

- name — zkratka charakterizující používaný operační systém; užitečná při psaní přenositelných programů
- system — provedení příkazu systému
- mkdir — vytvoření adresáře

  Př. >>>chdir(“c:\sw”)
  >>>getcwd()
  ‘c:\\sw’
- getcwd — zjistí současný pracovní adresář (z anglického **get c**urrent **w**orking **d**irectory)
- A další

re Umožňuje manipulaci s řetězci předepsanou regulárními výrazy, jaké se používají v systému Unix:

- search — hledej vzorek kdekoliv v řetězci
- match — hledej pouze od začátku řetězce
- findall — nalezne všechny výskyty vzorku v řetězci
- split — rozděl na podřetězce, které jsou odděleny zadaným vzorkem
- sub, subn — náhrada řetězců
- A další

math **Zpřístupňuje řadu matematických funkcí:**

- sin, cos, atd. — trigonometrické funkce
- log, log10 — přirozený a dekadický logaritmus
- ceil, floor — zaokrouhlení na celé číslo nahoru a dolů
- pi, e — konstanty
- …

time **Funkce pro práci s časem a datem:**

- time — vrací současný čas (vyjádřený v sekundách)
- gmtime — převod času v sekundách na UTC (tj. na čas v univerzálních časových souřadnicích — známější pod zkratkou GMT z anglického Greenwich Mean Time, tedy greenwichský [grinidžský] čas)
- localtime — převod do lokálního času (tj. posunutého vůči UTC o celé hodiny)
- mktime — opačná operace k localtime
- sleep — pozastaví běh programu na zadaný počet sekund
- …

random Generátory náhodných čísel :

- randint — generování náhodného čísla mezi dvěmi hranicemi (včetně)
- sample — generování náhodného podseznamu z jiného seznamu
- seed — počáteční nastavení klíče pro generování čísel
- …

Python – vstupy a výstupy, souborové objekty

```
f = open(jméno [, mod [, bufsize]])          # Otevření souboru, [ ] jsou metasymboly
                        # mod je r = read, w = write; bufsize = velikost buferu, moc se neužívá
f.read()                # přečte celý soubor a vrátí ho jako řetězec
f.read(n)               # přečte dalších n znaků, když jich zbývá méně – tak jich vrátí méně
f.readline()            # vrátí další řádku f nebo prázdný řetězec, když je f dočteno
f.readlines()           # přečte všechny řádky f a vrátí je jako seznam řetězců včetně ukončení
f.write(s)              # zapíše řetězec s do souboru f
f.writelines(L)         # zapíše seznam řetězců L do souboru f
f.seek(p [ , w])        # změní pozici: při w=0 na p, při w=1 jde o p dopředu, při w=2 na p odzadu
                         [ , w] je nepovinné
f.truncate([p])         # vymaže obsah souboru za pozicí p
f.tell()                # vrací současnou pozici v souboru
f.flush()               # vyprázdní buffer zkompletováním všech transakcí s f
f.isatty()              # predikát – je tento soubor terminál?
f.close()               # uzavře soubor f
```

Python – vstupy a výstupy

Př P3citacSlov.py Spustit Idle, File- Open- P3citacSlov.py, Run- Run Module

```
def pocetSlov(s):
    seznam = s.split()  # rozdeli retezec s na jednotliva slova a vrati jejich seznam
    return len(seznam)  # vrátíme počet prvků seznamu

vstup = open("jezek.txt", "r")  # otevre soubor pro cteni, alias pro open je file
                                # k otevření pro zápis použij argument “w”
                                # k otevření binárních použij argument “wb”, “rb”
                                # k otevření pro update použij +, např. “r+b”,  “r+”
                                # k otevření pro append  použij argument “a”
celkem = 0  # vytvoříme a vynulujeme proměnnou
for radek in vstup.readlines():  # vrátí seznam řádků textu, readlines(p) omezí na p bytů
      # soubor.read() přečte celý soubor a dodá ho jako string, možno i read(pbytů)
      # metoda readline() přečte 1 řádek končící \n
    celkem = celkem + pocetSlov(radek) # sečti počty za každý řádek
print "Soubor ma %d slov." % celkem  # tisk dle ridiciho retezce

vstup.close()                   # zavreni souboru
```

Následuje př.P31citacSloviZnaku.py . Lze spustit **a)** **Po spusteni v Idle na vyzvu napsat: jezek.txt**

Nebo b) v CMD C:\sw\Python25>Python d:\PGS\Python\P31citacSloviZnaku.py d:\PGS\Python\jezek.txt

Take b) v CMD D:\PGS\Python> C:\sw\Python25>Python P31citacSloviZnaku.py Python\jezek.txt

argument 0 argument 1

```
# -*- coding: cp1250 -*-
import sys
# jmeno souboru si vyžádáme od uživatele,
if len(sys.argv) != 2:
    jmenoSouboru = raw_input("Zadejte jmenosouboru: ") #vestavena fce pro cteni retezce
else:   # else ho zadame z prikazoveho radku tj. Moznost b)
    jmenoSouboru = sys.argv[1] #je to 2.argument prikazu vyvolani Pythonu
vstup = open(jmenoSouboru, "r")
# vytvoříme a znulujeme příslušné proměnné.
slov = 0
radku = 0
znaku = 0
for radek in vstup.readlines():
    radku = radku + 1
    # Řádek rozložíme na slova a spočítáme je.
    seznamSlov = radek.split()
    slov = slov + len(seznamSlov)
    # Počet znaků určíme z délky původního řádku,
znaku = znaku + len(radek)
print "%s ma %d radku, %d slov a %d znaku" % (jmenoSouboru, radku, slov, znaku)
vstup.close()
```

Python – vstupy a výstupy

Způsob vyvolání ze shellu Pythonu:

```
>>> import P31citacSloviZnaku
Zadejte jmeno souboru: d:\pgs\Python\jezek.txt
```

Nebo z příkazového řádku DOS

```
c:\Python25> d:\PGS\Python\ P31citacSloviZnaku.py  d:\pgs\Python\jezek.txt
```

dtto je `c:\Python25> python d:\PGS\Python\ P31citacSloviZnaku.py  d:\pgs\Python\jezek.txt`

Nebo spustíme z IDLE

Je-li čtený soubor v pracovním directory IDL, stačí
Zadejte jmeno souboru: jezek.txt

Další př. I/O možnosti:

-číst řádky lze i cyklování přes objekt typu soubor: `for line in file:  #napr. for line in vstup`
`print line  #  print line`

-zapisovat lze i metodou write(řetězec) na řetězec lze vše převést metodou str(něco)

-print vst.tell() tiskne pozici ve file

-vst.seek(10,1) změní aktuální pozici v souboru otevřeném v vst o 10 dál

-modul pickle má metody pro konverzi objektů na řetězce a opačně

pickle.dump(objekt, file_otevřeno_pro_zápis) zakleje objekt do souboru, který lze poslat po síti, nebo uložit v paměti jako perzistentní objekt

objekt = pickle.load(file_pro_čtení) vytvoří ze souboru opět objekt