PYTHON
VÝPISKY

00_Obsah

Použitá literatura:

[1] HARMS, Daryl; MCDONALD, Kenneth. *Začínáme programovat v jazyce Python* . [s.l.] : Computer Press, 2006. ISBN 978-80-251-2161-0.

01_Základy

Členění kódu:

bloková struktura programu je určena odsazením textu jeho zdrojového kódu pomocí tabulátoru

```
x = 1 + 2 + 3 \     # zalomení řádku příkazu
+ 4 + 5
```

Komentáře:

```
# toto je komentář
```

Proměnné a přiřazení:

Python je dynamicky (do 1 proměnné lze ukládat data různého typu), silně (nepodporuje implicitní typovou konverzi – převod řetězce na číslo) typovaný skriptovací jazyk

```
x = 5                       # přiřazení celého čísla do proměnné
x = 50000000000L            # přiřazení dlouhého celého čísla do proměnné
x = 5.5                     # přiřazení desetinného čísla do proměnné
x = 5e-2                    # přiřazení desetinného čísla do proměnné
x = 5+2j                    # přiřazení komplexního čísla do proměnné
x = "Ahoj"                  # přiřazení řetězce do proměnné
x = 'Ahoj'                  # přiřazení řetězce do proměnné
x = None                    # přiřazení prázdné hodnoty do proměnné
x,y = 5,10                  # přiřazení více proměnných
del x                       # zrušení proměnné
int(x)                      # explicitní převod proměnné na celé číslo
long(x)                     # explicitní převod na dlouhé celé číslo
float(x)                    # explicitní převod na desetinné číslo
global x                    # definice globální proměnné

x = """Nazdar "chlape",
jak se mas?"""
```

popis: trojité uvozovky umožňují zápis řetězce na více řádků, zápis apostrofů a uvozovek bez použití escape znaků

```
x=1; y=2
z = (x+y) / 2       # proměnná z = 1
z = (x+y) / 2.0     # proměnná z = 1.5
```

popis: aritmetika jazyka dodržuje pravidla jazyka C pro určování datových typů, výsledkem aritmetické operace s celými čísly je opět celé číslo, výsledkem operace s čísly s pohyblivou desetinnou čárkou je opět číslo s pohyblivou desetinnou čárkou, výsledkem operace celého a dlouhého celého čísla je celé číslo, při operaci s celými čísly může dojít k přetečení (vyvolá chybové hlášení)

Matematické funkce:

vestavěné: abs, divmod, cmp, coerce, float, hex, int, long, max, min, oct, pow, round

modul math: acos, asin, atan, atan2, ceil, cos, cosh, e, exp, fabs, floor, fmod, frexp, hypot, ldexp, log, log10, mod, pi, pow, sin, sinh, sqrt, tan, tanh

02_Seznamy a n-tice

Seznam (list):

```
x = [1, 2, 3]             # seznam
x = [2, "dva", [1,2,3]]   # seznam s prvky různých datových typů
len(x)                    # funkce která vrací délku seznamu
```

Indexování seznamu:

```
x    =    [    1,   2,   3,   4    ]
           ^    ^    ^    ^    ^
+index      0    1    2    3    4
-index     -4   -3   -2   -1
```

```
x[0]                # index prvního prvku seznamu
x[-1]               # index posledního prvku seznamu
x[0] = "Ahoj"       # přiřazení řetězce do prvního prvku seznamu
x[0:3]              # řez seznamu - první až třetí prvek seznamu
x[1:-1]             # řez seznamu - druhý až předposlední prvek
x[:3]               # řez seznamu – začátek až třetí prvek seznamu
x[2:]               # řez seznamu – třetí prvek seznamu až konec
x[:]                # řez seznamu – celý seznam
```

Modifikace seznamu:

```
y = x[:]                # vytvoření kopie seznamu řezem
y = x * 1               # vytvoření kopie seznamu replikací
y = x + []              # vytvoření kopie seznamu zřetězením

x[len(x):] = [5,6,7]    # přidá seznam na konec seznamu
x[:0] = [-1,0]          # přidá seznam na začátek seznamu
x[1:8] = []             # odstraní druhý až osmý prvek ze seznamu

del x[1]                # vymaže druhý prvek seznamu
x[1:2] = []             # vymaže druhý prvek seznamu
del x[1:3]              # vymaže druhý až třetí prvek seznamu
x[1:3] = []             # vymaže druhý až třetí prvek seznamu

x.remove(3)             # vyhledá a vymaže první instanci hodnoty

x.reverse()             # obrací uspořádání prvků v seznamu

x.append("Ahoj")        # připojí jeden prvek ke konci seznamu
x.insert(2, "Ahoj")     # vloží jeden prvek mezi 2. a 3. prvek
```

popis: funkce insert neumí používat záporné indexy

Třídění seznamu:

```
x.sort()            # setřídí seznam prvků libovolných datových typů

import bisect             # modul bisect
bisect.insort(x, 4)       # umístí novou položku do setříděného seznamu
bisect.bisect(x, 4)       # vrací index pozice kam se má hodnota vložit
```

uživatelské třídění – viz učebnice str. 65

Další operace se seznamy:

```
"čau" in ["čau","pa"]    # zjistí existenci prvku seznamu, vrací True
x=[1,2,3,4]; 5 in x      # zjistí existenci prvku seznamu, vrací False
5 not in [1,2,3,4]       # zjistí existenci prvku seznamu, vrací True

x = [1,2,3] + [4,5,6]    # zřetězení seznamu

x = [None] * 4           # inicializace seznamu o 4 prvcích
x = [1,2,3] * 2          # replikace seznamu

x=[3,7,0,-2,11]; min(x)  # vrací nejmenší prvek seznamu
max(["cau","zdar","pa"]) # vrací největší prvek seznamu

x=[3,7,0,-2,11]; x.index(7)   # vrací index daného prvku seznamu

x=[1,2,3,3,4,4]; x.count(3)   # vrací počet výskytů prvku v seznamu
```

Vnořené seznamy, matice a hluboké kopie:

```
x = [[0,1,2], [10,11,12], [20,21,22]]   # dvourozměrná matice
x[0]                                    # vrací [0,1,2]
x[2][1]                                 # vrací 21
```

hluboké kopie – viz učebnice str. 69

N-tice (tuple):

jsou podobné seznamům, ale nelze je modifikovat a slouží jako klíče slovníků

```
x = (1, 2, 3)             # n-tice
x = (2, "dva", [1,2,3])   # n-tice s prvky různých datových typů
y = x[:]                  # vytvoření kopie n-tice řezem
y = x * 1                 # vytvoření kopie n-tice replikací
y = x + ()                # vytvoření kopie n-tice zřetězením
x = (5,)                  # vytvoření jednoprvkové n-tice
x = ()                    # vytvoření prázdné n-tice

x[2]                      # index třetího prvku n-tice
x[1:]                     # řez n-tice – druhý prvek seznamu až konec
len(x)                    # funkce která vrací délku n-tice
min(x)                    # vrací nejmenší prvek n-tice
max(x)                    # vrací největší prvek n-tice
5 in x                    # zjistí existenci prvku n-tice
x = (1,2,3) + (4,5,6)     # zřetězení n-tice
x = (1,2,3) * 2           # replikace n-tice

(jedna,dve,tri)=(1,2,3)   # přiřazení hodnot n-tici
jedna,dve,tri=1,2,3       # přiřazení hodnot n-tici
prom1,prom2=prom2,prom1   # záměna hodnot dvou proměnných
```

popis: skládání a rozkládání lze provádět také u seznamu

Převody mezi seznamy a n-ticemi:

```
list((1, 2, 3, 4))        # převod n-tice na seznam
tuple([1, 2, 3, 4])       # převod seznamu na n-tici
```

popis: funkce list lze použít pro rozložení řetězce na znaky, např. `list("Ahoj")`

03_Řetězce

Řetězce jakožto sekvence znaků:

řetězce nelze modifikovat

```
x = "Ahoj"      # přiřazení řetězce do proměnné
x = 'Ahoj'      # přiřazení řetězce do proměnné

x[0]            # index prvního znaku řetězce
x[-1]           # index posledního znaku řetězce
x[1:]           # řez řetězce - druhý znak řetězce až konec

len("Ahoj")     # vrací počet znaků v řetězci

x = "Nazdar " + "chlape"      # zřetězení řetězců
x = "1" * 8                   # replikace řetězce

"zdar" in "Nazdar chlape"     # zjistí existenci podřetězce v řetězci
```

Escape sekvence:

escape znaky: \\ (backslash), \' (apostrofy), \" (uvozovky), \n (nový řádek), \t (tabulátor), \a (zvonek), \b (backspace), \f (nová stránka), \r (návrat vozíku), \v (svislý tabulátor)

numerické escape znaky: *oktalové_číslo* (vyjádření znaku v osmičkové soustavě), \x*hexadec_číslo* (vyjádření znaku v hexadecimální soustavě), např. `"m" = "\155" = "\x6d"`

Tisk vs. vyhodnocení:

```
"a\tb"              # vyhodnocení, výstup: 'a\tb'
print "a\tb"        # tisk, výstup: a   b
print "Ahoj\n",     # čárka na konci zabrání příkazu vytisknout \n
```

Modul string:

```
import string                              # modul string

string.join(["Nazdar","chlape"])           # spojí řetězce do jednoho
string.join(["Nazdar","chlape"],"::")      # oddělovačem řetězců je ::

string.split("Nazdar chlape")              # rozdělí řetězec na části
string.split("Nazdar::chlape","::")        # oddělovačem řetězců je ::
string.split("1 2 3 4 5",None,2)           # rozdělí řetězec na 3 části

string.atoi("123")       # převod řetězce na celé číslo
string.atoi("123",16)    # převod řetězce na celé číslo v hexadec. s.
string.atol("123")       # převod řetězce na dlouhé celé číslo
string.atof("123.456")   # převod řetězce na desetinné číslo
```

popis: převod řetězce na číslo lze provést také pomocí vestavěných funkcí int(), long(), float()

```
string.strip("\tNazdar chlape ")  # odstraní bílé znaky na krajích
string.lstrip("\tNazdar chlape ") # odstraní bílé znaky na začátku
string.rstrip("\tNazdar chlape ") # odstraní bílé znaky na konci
```

popis: bílé znaky jsou definovány v konstantě string.whitespace

```
string.find("Mississippi","ss")    # vrací index 1. nalez. podřetězce
string.find("Mississippi","ss",3)      # vyhlédává až od 3. pozice
string.find("Mississippi","ss",0,3)    # vyhlédává od 0. do 3. pozice
string.rfind("Mississippi","ss")   # vyhledává odzadu
string.index("Mississippi","ss")   # vrací index 1. nalez. podřetězce
string.rindex("Mississippi","ss")  # vyhledává odzadu
string.count("Mississippi","ss")   # vrací počet výskytů podřetězce
```

popis: funkce rfind(), index(), rindex(), count() používají stejné parametry jako funkce find(); funkce index() a rindex() fungují stejně jako funkce find() a rfind() s tím rozdílem, že když funkce nenalezne žádný podřetězec, nevrací hodnotu -1, ale vyvolá výjimku; funkce count() vrací počet výskytů nepřekrývajících se podřetězců

```
string.replace("Mississippi","ss","...")     # náhrada podřetězce

mapa=string.maketrans("~^()","!&[]")
string.translate("~x^(y%z)",mapa)      # vrací '!x&[y%z]'
```

popis: překlad znaků na základě překladové mapy

```
string.lower()      # převede znaky na malá písmena
string.upper()      # převede znaky na velká písmena
string.capitalize() # převede první písmeno řetězce na velké
string.capwords()   # převede znaky všech slov řetězce na velké
string.swapcase()   # zamění velká a malá písmena
string.expandtabs() # nahradí tabulátory za odpovídající počet mezer
string.ljust()      # doplní řetězec mezerami do požadované délky
string.rjust()      # doplní řetězec mezerami do požadované délky
string.center()     # doplní řetězec mezerami do požadované délky
string.zfill()      # doplní číselný řetězec zleva
```

Převod objektů na řetězce:

```
repr([1,2,3])       # převod seznamu na řetězec
`[1,2,3]`           # převod seznamu na řetězec
str([1,2,3])        # převod seznamu na řetězec
```

Formátování řetězců:

```
"%s je %s" % ("Svickova","vynikajici")      # řetězcový modulus
x=[1,2]; "%s obsahuje %s" % ("Seznam",x)    # řetězcový modulus
"Pi je %-6.2f" % 3.14159                    # formátovací sekvence
```

04_Slovníky (hashovací tabulky)

Slovník:

```
x={}                # slovník
x[0]="Nazdar"       # přiřazení řetězce do prvního prvku slovníku
x["pi"]=3.14        # přiřazení čísla do slovníku pomocí klíče
x["ahoj"]="hello"   # přiřazení řetězce do slovníku pomocí klíče
slovnik={"cervena":"red","modra":"blue"}    # definice slovníku

x[0] + " chlape"          # přístup k prvku slovníku
print x["ahoj"],"boy"     # přístup k prvku slovníku

len(slovnik)              # vrací počet prvků ve slovníku
slovnik.keys()            # vrací seznam všech klíčů ve slovníku
slovnik.values()          # vrací seznam všech hodnot ve slovníku
slovnik.items()           # vrací n-tice všech hodnot a klíčů slovníku
del slovnik["modra"]      # odstraní položku ze slovníku
slovnik.has_key("bila")   # testuje zda ve slovníku existuje daný klíč
slovnik.get("bila","NA") # vrací hodnotu pro klíč nebo defin. hodnotu
x=slovnik.copy()          # vytvoří kopii slovníku
x.update(slovnik)         # aktualizuje slovník
```

seznamy nelze použít jako klíče slovníku, n-tice ano

05_Řízení běhu programu

Cyklus while:

```
while x<10:
     x+=1
```

popis: uvnitř cyklu while lze použít příkazy `break` a `continue` a také `else`

Podmínka if:

```
if x==0:
     x=1
elif x==1:
     x=0
else:
     x=-1
```

Cyklus for:

```
seznam=[1.0, 2.0, 3.0]
for x in seznam:
     print 1/x

seznam=[1,2,-7,4,9,-5,4]
for x in range(len(seznam)):      # fce range prochází indexy
     if seznam[x] < 0:
          print "Zaporne cislo nalezeno na indexu",x

range(3,7)     # počáteční a koncový index, např. [4,5,6,7]
range(0,7,2)   # počát. a koncový index s přírůstkem, např. [0,2,4,6]
range(4,0,-1)  # počát. a koncový index s přírůstkem, např. [4,3,2,1]
```

popis: funkce xrange funguje stejně jako range, ale je pomalejší a nevytváří seznam takže zabírá méně paměti

Logické operátory:

<	menší
>	větší
<=	menší rovno
>=	větší rovno
==	rovno
!=	nerovno
<>	nerovno
and	a zároveň
or	nebo
not	negace
in	testuje přítomnost prvku v sekvenci (seznamu, n-tici, řetězci, slovníku)
is	testuje zda jsou dva objekty stejné

popis: pro řízení priority operátorů lze použít (); intervaly lze psát i takto: `if 0 < x < 10:`

hodnotu false představují: `0, 0.0, 0L, 0+0j, "", [], {}, None`

06_Funkce a procedury

Definice funkcí a procedur:

```
def faktorial(n):        # definice funkce
     r=1
     while n>0:
          r=r*n
          n=n-1
     return r
faktorial:               # volání funkce
```

procedura nevrací žádnou hodnotu (resp. vrací None) narozdíl od funkce

```
lambda x, y: x+y         # definice funkce na místě
```

Předávání argumentů jménem parametru:

```
def mocnina(x,y=2):      # definice implicitní hodnoty funkce
     r=1
     while y>0:
          r=r*x
          y=y-1
     return r

mocnina(3,3)             # volání funkce
mocnina(3)               # volání funkce s využitím implicitní hod.
mocnina(y=2,x=3)         # volání funkce s předáním argum. jménem
```

Proměnlivý počet argumentů:

```
def maximum(*cisla):     # argumenty se načtou do n-tice
     if len(cisla) == 0:
          return(None)
     else:
          max = cisla[0]
          for n in cisla[1:]:
               if n > max: max = n
          return max

maximum(3, 2, 8, 1)      # volání funkce s proměnlivým počtem arg.
```

```
def funkce(x, y, **slovnik):  # argumenty se načtou do slovníku
     soucet=0
     for k in slovnik.keys():
          soucet += slovnik[k]
     print "Soucet hodnot v parametru slovik je:",soucet

funkce(1,2,a=3,b=4,c=5,d=6)
```

07_Moduly

Moduly:

modul je samostatný textový soubor napsaný v pythonu (např. mujmodul.py) nebo c/c++ obsahující příbuzné funkce a konstanty

```
import mujmodul                 # importuje modul
mujmodul.funkce()               # volání funkce z modulu
from mujmodul import a(),b()    # importuje funkce z modulu
from mujmodul import *          # importuje všechny objekty z modulu
reload(mujmodul)                # znovunačtení modulu po změně modulu
```

Chráněná jména v modulech:

při importu objektů z modulu pomocí příkazu `from mujmodul import *` se neimportují objekty začínající znakem _ (např. _var, _funkce()), tyto objekty je nutné importovat zvlášť příkazem:

```
from mujmodul import _var
import mujmodul; mujmodul._var
```

Pravidla rozsahu:

viz učebnice str. 130

08_Souborový system

Cesty a jejich popis:

```
import os
os.getcwd()              # aktuální pracovní adresář
os.listdir(os.curdir)    # vrací seznam všech souborů v pracovním adr.
os.chdir("slozka")       # změna aktuálního adresáře

os.path.join("bin","utils")   # vytvoření cesty k souboru
os.path.split(cesta)          # rozdělí cestu za posledním lomítkem
os.path.basename(cesta)  # vrací poslední část cesty (bázi)
os.path.dirname(cesta)        # vrací celou cestu bez poslední části
os.path.splitext(cesta)  # vrací příponu souboru včetně .

os.curdir      # aktuální adresář
os.pardir      # nadřazený adresář
os.name        # identifikace OS
sys.platform   # identifikace platformy
os.environ     # slovník s proměnnými prostředí
```

Informace o souborech:

```
os.path.exists(cesta)         # zjistí zda daná cesta existuje
os.path.isfile(cesta)         # zjistí zda je cesta souborem
os.path.isdir(cesta)          # zjistí zda je cesta adresářem
os.path.islink(cesta)         # zjistí zda je cesta odkazem (jen UNIX)
os.path.ismount(cesta)        # zjistí zda je cesta mounted (jen UNIX)
os.path.samefile(cesta1,cesta2)# zda cesty ukazují na 1 soubor
os.path.isabs(cesta)          # zjistí zda je cesta absolutní
os.path.getsize(cesta)        # vrací velikost cesty
os.path.getmtime(cesta)       # vrací poslední datum změny cesty
os.path.getatime(cesta)       # vrací poslední čas přístupu k cestě
```

Další operace:

```
import glob
glob.glob("*")      # vypíše seznam všech souborů dle hvězdičkové
                      konvence unixu: * ? [a-z,A-Z]

os.rename("novy","stary")     # přejmenování a přesunutí souboru
os.remove("soubor")           # smazání souboru
os.mkdir("adresar")           # vytvoření adresáře
os.makedirs(cesta)            # vytvoření více vnořených adresářů
os.rmdir("adresar")           # smazání prázdného adresáře
```

```
os.path.walk(adresar,fce,arg_fce) # průchod adresářovou strukturou
```

```
import os
def tisk(arg, adresar, jmena):
     print adresar, len(jmena)
os.path.walk(os.getcwd(), tisk, None)
```

popis: projde aktuálním adresářem a jeho podadresáři a vypíše počet souborů a adresářů

modul shutil obsahuje další funkce pro práci se soubory včetně funkce rmtree pro vymazání adresářové struktury včetně souborů

09_Čtení a zápis do souboru

Otevření souboru:

```
soubor = open("text.txt","r")      # otevření souboru pro čtení
soubor = open("text.txt","w")      # otevření souboru pro přepis
soubor = open("text.txt","a")      # otevření souboru pro zápis
soubor = open("text.txt","rb")     # otevření bin. souboru pro čtení
soubor = open("text.txt","wb")     # otevření bin. souboru pro přepis
soubor.close                       # uzavření souboru
```

Funkce pro čtení a zápis:

```
radek = soubor.readline()          # přečtení řádky ze souboru
radky = soubor.readlines()         # uloží všechny řádky do seznamu
hlavicka = soubor.read(4)          # přečte první 4 bajty souboru
```

popis: funkce readline() vrací "", pokud už nejsou v souboru žádná data

```
soubor.write("Nazdar!\n")          # zápis řetězce do souboru
soubor.writelines(seznam)          # zápis seznamu řetězců do souboru
```

Funkce vstupu a výstupu na obrazovku:

```
x = raw_input("Zadej jméno:")      # čtení řetězce z obrazovky
x = int(raw_input("Zadej číslo:"))
x = input("Zadej číslo: ")         # čtení dat z obrazovky
```

popis: čtení a zápis na standardní výstup lze také provést pomocí objektů sys.stdin, sys.stdout, sys.stderr, které mají funkce read, readline, readlines, resp. write, writelines; přesměrování viz učebnice str. 156

Modul Struct:

```
import struct
format_zaznamu = "hd4s"  # h: short int, d: double float, s: string
     velikost_zaznamu = struct.calcsize(format_zaznamu)
     vysledny_seznam = []
     vstup = open("data","rb")
     while 1:
          zaznam = vstup.read(velikost_zaznamu)
          if zaznam == "":
               vstup.close()
               break
          vysledny_seznam.append(struct.unpack(format_zaznamu,zaznam))
```

Nakládání objektů do souboru:

```
import cPickle
a = 1
soubor = open("stav","w")
cPickle.dump(a,soubor)        # uložení stavu proměnné do souboru
soubor.close()

soubor = open("stav","r")
cPickle.load(soubor)          # načtení stavu proměnné
soubor.close()



import shelve
adresar = shelve.open("adresy")
adresar["00001"]=["Petr Svoboda","607555111","Brno"]
adresar["00002"]=["Jan Novotny","721567323","Praha"]
adresar.close()

import shelve
adresar = shelve.open("adresy")
adresar["00001"]
adresar.close()
```

10_Výjimky

Zachycení výjimek:

```
try:
     tělo
except vyjimkaTyp1, var1:
     vyjimkaKod1
except vyjimkaTyp2, var2:
     vyjimkaKod2
except:
     standardniKodVyjimky
else:
     jineTelo

try:
     tělo
finally:       # provede se vždy
     tělo
```

Definování nových výjimek:

```
class MojeChyba(Exception)
     pass

try:
     raise MojeChyba, "Info o chybe"
except MojeChyba, chyba
     print "Situace", chyba
```

více o výjimkách – viz učebnice str. 167

11_Skripty

Vytvoření a spouštění skriptu:

```
def main():
     print "Nazdar chlape!"
main()
```

popis: skript spustíme příkazem python skript.py, v UNIXu lze spustit skript přímo přidáním kódu #! /usr/bin/env python

Parametry příkazové řádky:

```
import sys
def main():
     print sys.argv
main()
```

```
import getopt, sys
def main():
     (volby,argumenty) = getopt.getopt(sys.argv[1:],"f:vx:y:z:")
     print "volby:",volby
     print "argumenty:",argumenty
main()
```

popis: python skript.py –x100 –v –y50 –f soubor arg1 arg2; dvojtečka znamená že přepínač vyžaduje parametr

Modul fileinput:

```
import fileinput
def main():
     for radek in fileinput():
          if radek[:2] != "##":
               print radek
main()
```

popis: python skript.py soubor1 soubor2; vypíše řádky z daných souborů, které nemají na začátku ##; více o modulu fileinput viz učebnice str. 186

12_Třídy a OOP

Třídy a instance:

```
class Kruh:
     pi = 3.14159                         # proměnná třídy
     def __init__(self,polomer=1):        # konstruktor
          self.polomer = polomer          # proměnná instance
          self.__prom = "prom"            # soukromá proměnná instance
     def plocha(self):                    # metoda
          return self.polomer * self.polomer * self.__class__.pi
     def __del__(self):                   # destruktor
          print "destruktor"
print instance.__class__.pi               # proměnná instance
print Kruh.pi                             # proměnná třídy
instance = Kruh()                         # vytvoření instance (objektu)
print instance.plocha()                   # volání metody
instance = Kruh(3)                        # vytvoření instance (objektu)
print instance.plocha()                   # volání metody
instance.polomer = 5                      # proměnná instance
print instance.plocha()                   # volání metody
```

Dědičnost:

```
class Tvar:
     def __init__(self,x,y):
          self.x = x
          self.y = y
class Ctverec(Tvar):
     def __init__(self,strana=1,x=0,y=0):
          Tvar.__init__(self,x,y)
          self.strana = strana
class Kruh(Tvar):
     def __init__(self,r=1,x=0,y=0):
          Tvar.__init__(self,x,y)
          self.polomer = r
```

13_Regulární výrazy

Regulární výraz:

```
import re
regv = re.compile("ahoj")
pocet = 0
soubor = open("text.txt","r")
for radek in soubor.readlines():
     if regv.search(radek):
          pocet = pocet + 1
soubor.close()
print pocet



("ahoj|Ahoj")
("(a|A)hoj")
("[aA]hoj")
print r"C:\"        # neupravené řetězce (ruší význam spec. znaků)
```