

# Řetězce a seznamy (a kryptografické odbočky)

IB111 Úvod do programování skrze Python

2013

# Rozcvička: šifry

① C S A R B V  
E K T E O A

② C S B U J T M B W B

③ A J L B N O C E

# Transpoziční šifry

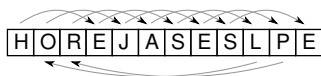
pozpátku



trojice pozpátku



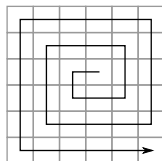
ob tři



dopředu dozadu

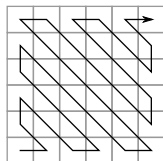


šnek



L	B	A	K	I	N
I	C	S	E	J	B
Z	H	O	P	D	Y
K	O	K	L	A	R
O	V	A	N	Y	U
U	H	R	A	Z	E

cik-cak



N	I	O	U	Z	E
H	B	K	K	H	A
C	O	Y	A	Z	R
L	S	V	R	B	I
K	A	E	A	U	L
P	O	D	J	N	Y

# Substituční šifry

## Jednoduchá substituce - posun o 3 pozice

	K	O	Z	A
	↓	↓	↓	↓
	10	14	25	0
+3	↓	↓	↓	↓
	13	17	2	3
	↓	↓	↓	↓
	N	R	C	D

## Substituce podle hesla

HLEDEJPODLIPOU	H → 7	+ →	25 → Z
SLONSLONSLONSL	S → 18		
ZWSQWUDBVWWCGF			

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

# Řetězce a znaky – ukázky operací

```
"kos" * 3  
"petr" + "klic"  
text = "velbloud"  
len(text)  
text[0]  
text[2]  
text[-1]  
ord('b')  
chr(99)
```

str() – explicitní přetypování na řetězec

# ASCII tabulka, ord, chr

1 r	33 !	65 A	97 a	129 Ì	161 ÿ	193 Á	225 á
2 1	34 "	66 B	98 b	130 ,	162 ð	194 Â	226 â
3 1	35 #	67 C	99 c	131 f	163 é	195 Ã	227 ã
4 J	36 \$	68 D	100 d	132 .	164 ñ	196 Ä	228 ä
5	37 %	69 E	101 e	133 ...	165 ¥	197 Å	229 å
6 -	38 &	70 F	102 f	134 †	166 †	198 Æ	230 æ
7 •	39 '	71 G	103 g	135 ‡	167 ‡	199 Ç	231 ç
8 □	40 (	72 H	104 h	136 ^	168 ~	200 È	232 è
9	41 )	73 I	105 i	137 %	169 ©	201 É	233 é
10	42 *	74 J	106 j	138 Š	170 º	202 Ê	234 ê
11 ¢	43 +	75 K	107 k	139 <	171 <	203 Ë	235 ë
12 □	44 ,	76 L	108 l	140 Œ	172 ¬	204 Ì	236 ì
13	45 -	77 M	109 m	141 Ì	173 -	205 Í	237 í
14 #	46 .	78 N	110 n	142 Ž	174 ®	206 Î	238 î
15 #	47 /	79 O	111 o	143 Ì	175 ~	207 Ï	239 ï
16 †	48 0	80 P	112 p	144 Ì	176 °	208 Ð	240 ð
17 ◀	49 1	81 Q	113 q	145 '	177 ±	209 Ñ	241 ñ
18 ↓	50 2	82 R	114 r	146 "	178 º	210 Ò	242 ò
19 !!	51 3	83 S	115 s	147 "	179 º	211 Ó	243 ó
20 ¶	52 4	84 T	116 t	148 "	180 ´	212 Ô	244 ô
21 ↓	53 5	85 U	117 u	149 •	181 µ	213 Õ	245 õ
22 T	54 6	86 V	118 v	150 -	182 ¶	214 Ö	246 ö
23 †	55 7	87 W	119 w	151 —	183 -	215 ×	247 ×
24 †	56 8	88 X	120 x	152 "™	184 .	216 Ø	248 ø
25 †	57 9	89 Y	121 y	153 ™	185 '	217 Ù	249 ù
26 →	58 :	90 Z	122 z	154 §	186 °	218 Ú	250 ú
27 +	59 ;	91 [	123 {	155 >	187 >	219 Û	251 û
28	60 <	92 \	124	156 œ	188 ¼	220 Ü	252 ü
29	61 =	93 ]	125 }	157 Ì	189 ½	221 Ý	253 ý
30	62 >	94 ^	126 ~	158 ž	190 ¾	222 Þ	254 þ
31	63 ?	95 _	127 □	159 Ÿ	191 ÷	223 ß	255 ÿ
32	64 @	96 `	128 €	160	192 À	224 à	

# Řetězce – pokročilejší indexování

```
text = "velbloud"  
text[:3]      # první 3 znaky  
text[3:]      # od 3 znaku dále  
text[1:8:2]   # od 2. znaku po 7. krok po 2  
text[::3]     # od začátku do konce po 3
```

# Řetězce – změny

- neměnitelné (immutable) – rozdíl oproti seznamům a řetězcům v některých jiných jazycích
- změna znaku – vytvoříme nový řetězec

```
text = "kopec"  
text[2] = "n" # chyba  
text = text[:2] + "n" + text[3:]
```



# Řetězce: další operace

```
text = "bezi liska k Taboru"  
print text.upper()  
print text.lower()  
print text.capitalize()  
print text.rjust(30)  
print "X",text.center(30),"X"  
print text.replace("liska","jezek")
```

... a mnoho dalších, více viz dokumentace, později.

# Příklad: Transpozice (rozcvička I)

- přepis textu po sloupcích:
- příklad vstupu a výstupu:
  - C E S K A T R E B O V A
  - C S A R B V
  - E K T E O A

# Transpozice (rozcvička I)

```
def sifra_po_sloupcich(text,n):  
    for i in range(n):  
        for j in range(len(text) / n + 1):  
            pozice = j * n + i  
            if pozice < len(text):  
                print text[pozice],  
        print
```

# Transpozice (rozcvička I), kratší varianta

```
def sifra_po_sloupcich(text,n):  
    for i in range(n):  
        print text[i::n]
```

# Caesarova šifra (rozcvička II)

- substituční šifra – posun v abecedě
- vstup: text, posun
- výstup: zašifrovaných text
- BRATISLAVA, 1 → CSBUJTMBWB

# Caesarova šifra – řešení

```
def caesarova_sifra(text, n):  
    vystup = ""  
    text = text.upper()  
    for i in range(len(text)):  
        if text[i] == ' ': vystup = vystup + ' '  
        else:  
            c = ord(text[i]) + n  
            if (c > ord('Z')): c = c - 26  
            vystup = vystup + chr(c)  
    return vystup
```

# Caesarova šifra – rozlomení

- máme text zašifrovaný Caesarovou šifrou (s neznámým posunem)
- jak text dešifrujeme?
- příklad: MPKTDVLDVLMZCF

# Caesarova šifra – rozlomení

- máme text zašifrovaný Caesarovou šifrou (s neznámým posunem)
- jak text dešifrujeme?
- příklad: MPKTDVLDVLMZCF
- jak to udělat, aby program vrátil jen jednoho kandidáta?



# Caesarova šifra – rozlomení

$k$	Kandidát	$b_s$	$b_f$	$k$	Kandidát	$b_s$	$b_f$
0	MPKWTVDVLVELMZCF	0	21	13	ZCXGJGQIYIRYZMPS	0	-13
1	NQLUXUEWMWFMNADG	13	0	14	ADYHKHRJZJSZANQT	0	16
2	ORMVYVFXNXGNOBEH	24	9	15	BEZILISKAKTABORU	<b>67</b>	<b>59</b>
3	PSNWZWGYOYHOPCFI	5	-3	16	CFAJMJTLBLUBCPSV	0	11
4	QTOXAXHZPZIPQDGJ	10	-6	17	DGBKNKUMCMVCDQTW	5	-4
5	RUPYBYIAQAJQREHK	0	9	18	EHCLOLVNDNWDERUX	17	31
6	SVQZCZJBRBKRSFIL	0	3	19	FIDMPMWEOEXEFSVY	5	22
7	TWRADAKCSCSLSTGJM	32	26	20	GJENQNXPFPYFGTWZ	4	-23
8	UXSBEBLDTDMTUHKN	0	24	21	HKFOROYQGQZGHUXA	16	-17
9	VYTFCFCMEUENUVILO	11	46	22	ILGPSPZRHRAHIVYB	28	18
10	WZUDGDNFVFOVWJMP	0	-6	23	JMHQTQASISBIJWZC	9	0
11	XAVEHEOGWGPWXKNQ	5	-2	24	KNIRURBTJTCJKXAD	5	24
12	YBWFIFPHXHQXYLOR	0	-28	25	LOJSVSCUKUDKLYBE	4	29

# Vigenèrova šifra

- substituce podle hesla – „sčítáme“ zprávu a heslo
- vhodné cvičení
- rozlomení Vigenèrovovy šifry?

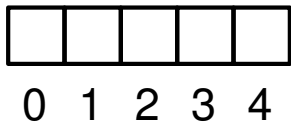
# Seznamy (pole) – motivace

- řazení studentů podle bodů na písemce
- reprezentace herního plánu (piškvorky, šachy)
- frekvence písmen v textu

# Frekvenční analýza nevhodně

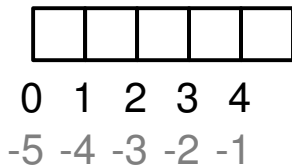
```
def frekvencni_analyza(text):  
    frekA = 0  
    frekB = 0  
    frekC = 0  
    for pismeno in text:  
        if pismeno == 'A':  
            frekA += 1  
        elif pismeno == 'B':  
            frekB += 1  
        elif pismeno == 'C':  
            frekC += 1  
    print 'A', frekA  
    print 'B', frekB  
    print 'C', frekC
```

# Seznamy (pole)



- „více položek za sebou v pevném pořadí“
- indexováno od nuly!
- základní koncept dostupný ve všech jazycích, běžně „pole“ (array), položky stejného typu, pevně daná délka
- seznamy v Pythonu – obecnější
- Python a pole – knihovna NumPy (nad rámec IB111)

# Seznamy v Pythonu



- seznam (list), n-tice (tuple)
- položky mohou být různého typu
- variabilní délka
- indexování i od konce (pomocí záporných čísel)

# Seznamy: použití v Pythonu

```
s = []          # deklarace prázdného seznamu
s = [3, 4, 1, 8 ]
s[2]           # indexace prvku, s[2] = 1
s[-1]         # indexace od konce, s[-1] = 8
s[2] = 15     # změna prvku
s.append(6)   # přidání prvku
s[1:4]        # indexace intervalu, s[1:4] = [4, 15, 8]
len(s)        # délka seznamu, len(s) = 5
t = [ 3, "pes", [2, 7], -8.3 ]
              # seznam může obsahovat různé typy
```

list() – přetypování na seznam

# Python: seznamy a cyklus for

- cyklus for – přes prvky seznamu
- range – vrací seznam čísel
- typické použití: `for i in range(n):`
- ale můžeme třeba i:
  - `for zvire in ["pes", "kocka", "prase"]:` ...
  - `for pismeno in "velbloud":` ...



# Objekty, hodnoty, aliasy

a = [1, 2, 3]  
b = [1, 2, 3] nebo b = a[:]

a → [1, 2, 3]  
b → [1, 2, 3]

a = [1, 2, 3]  
b = a

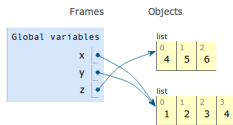
a → [1, 2, 3]  
b ↗ [1, 2, 3]

- parametry funkcí – pouze volání hodnotou (na rozdíl např. od Pascalu: volání hodnotou a odkazem)
- měnitelné objekty (např. seznam) však funkce může měnit
- n-tice (tuples) – neměnitelná varianta seznamů
- více na cvičeních, později

# Vizualizace běhu programu

<http://www.pythontutor.com/>

```
1 x = [1, 2, 3]
2 y = [4, 5, 6]
3 z = y
4 y = x
5 x = z
6
7 x = [1, 2, 3] # a different [1, 2, 3] list!
8 y = x
9 x.append(4)
10 y.append(5)
11 z = [1, 2, 3, 4, 5] # a different list!
12 x.append(6)
13 y.append(7)
14 y = "hello"
--
```



vhodné např. pokud je nejasný některý z příkladů ve slidech

## Příklad: výpočet průměrné hodnoty

```
def prumer1(seznam):  
    soucet = 0.0  
    for i in range(len(seznam)):  
        soucet += seznam[i]  
    return soucet / len(seznam)
```

```
def prumer2(seznam):  
    soucet = 0.0  
    for x in seznam:  
        soucet += x  
    return soucet / len(seznam)
```

```
def prumer3(seznam):  
    return float(sum(seznam)) / len(seznam)
```

# Ilustrace práce se seznamem

```
def seznam_delitelu(n):  
    delitele = []  
    for i in range(1, n+1):  
        if n % i == 0:  
            delitele.append(i)  
    return delitele  
  
delitele24 = seznam_delitelu(24)  
print delitele24  
print len(delitele24)  
for x in delitele24: print x**2,
```

# Frekvenční analýza nevhodně

```
def frekvencni_analyza(text):  
    frekA = 0  
    frekB = 0  
    frekC = 0  
    for pismeno in text:  
        if pismeno == 'A':  
            frekA += 1  
        elif pismeno == 'B':  
            frekB += 1  
        elif pismeno == 'C':  
            frekC += 1  
    print 'A', frekA  
    print 'B', frekB  
    print 'C', frekC
```

# Frekvenční analýza lépe

```
def frekvencni_analyza(text):  
    frekvence = [ 0 for i in range(26) ]  
    for pismeno in text:  
        if ord(pismeno) >= ord('A') and\  
            ord(pismeno) <= ord('Z'):  
            frekvence[ord(pismeno) - ord('A')] += 1  
    for i in range(26):  
        if frekvence[i] != 0:  
            print chr(ord('A')+i), frekvence[i]
```

## Simulace volebního průzkumu – nevhodné řešení

```
def pruzkum(vzorek, pref1, pref2, pref3):  
    pocet1 = 0  
    pocet2 = 0  
    pocet3 = 0  
    for i in range(vzorek):  
        r = random.randint(1,100)  
        if r <= pref1: pocet1 += 1  
        elif r <= pref1 + pref2: pocet2 += 1  
        elif r <= pref1 + pref2 + pref3: pocet3 += 1  
    print "Strana 1:", 100.0 * pocet1 / vzorek  
    print "Strana 2:", 100.0 * pocet2 / vzorek  
    print "Strana 3:", 100.0 * pocet3 / vzorek
```

## Simulace volebního průzkumu – lepší řešení

```
def pruzkum(vzorek, pref):
    n = len(pref)
    pocet = [ 0 for i in range(n) ]
    for _ in range(vzorek):
        r = random.randint(1,100)
        for i in range(n):
            if sum(pref[:i]) < r <= sum(pref[:i+1]):
                pocet[i] += 1
    for i in range(n):
        print "Strana", i+1, 100.0 * pocet[i] / vzorek
```

Toto řešení má stále nedostatky (po stránce funkčnosti) – zkuste dále vylepšit.



# Převod do Morseovy abecedy

- vstup: řetězec
- výstup: zápis v Morseově abecedě
- příklad: PES  $\rightarrow$  .-- . | . | . . .

# Převod do Morseovy abecedy nevhodně

```
def prevod_morse(text):  
    vystup = ''  
    for i in range(len(text)):  
        if text[i] == 'A': vystup += '.-|'  
        elif text[i] == 'B': vystup += '-...|'  
        elif text[i] == 'C': vystup += '-.-|'  
        elif text[i] == 'D': vystup += '-..|'  
        # atd  
    return vystup
```

# Převod do Morseovy abecedy: využití seznamu

```
morse = ['.-.', '-...-', '-.-.', '-..'] # atd
```

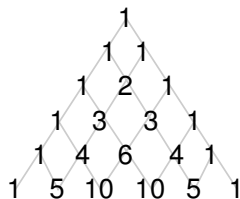
```
def prevod_morse(text):  
    vystup = ''  
    for i in range(len(text)):  
        if ord('A') <= ord(text[i]) <= ord('Z'):  
            c = ord(text[i]) - ord('A')  
            vystup += morse[c] + '|'  
    return vystup
```

(ještě lepší řešení: využití slovníku)

# Převod z Morseovy abecedy

```
def najdi_pismeno(sekvence):  
    for i in range(len(morse)):  
        if morse[i] == sekvence:  
            return chr(ord('A') + i)  
    return '?'  
  
def prevod_z_morse(zprava):  
    vystup = ''  
    sekvence = ''  
    for znak in zprava:  
        if znak == '|':  
            vystup += najdi_pismeno(sekvence)  
            sekvence = ''  
        else:  
            sekvence += znak  
    return vystup
```

# Pascalův trojúhelník



$$\begin{array}{c} \binom{0}{0} \\ \binom{1}{0} \quad \binom{1}{1} \\ \binom{2}{0} \quad \binom{2}{1} \quad \binom{2}{2} \\ \binom{3}{0} \quad \binom{3}{1} \quad \binom{3}{2} \quad \binom{3}{3} \end{array}$$

Explicitní vzorec

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

Rekurzivní vztah

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

# Pascalův trojúhelník

```
def pascaluv_trojuhelnik(n):  
    aktualni_radek = [ 1 ]  
    for i in range(n):  
        for x in aktualni_radek:  
            print x,  
        print  
        dalsi_radek = [ 1 ]  
        for i in range(len(aktualni_radek)-1):  
            dalsi_radek.append(aktualni_radek[i] +\  
                               aktualni_radek[i+1])  
        dalsi_radek.append(1)  
        aktualni_radek = dalsi_radek
```

- dělitelné jen 1 a sebou samým
- předmět zájmu matematiků od pradávna, cca od 70. let i důležité aplikace (moderní kryptologie)
- problémy s prvočísly:
  - výpis (počet) prvočísel v intervalu
  - test prvočíselnosti
  - rozklad na prvočísla (hledání dělitelů)

# Výpis prvočísel přímočaře

```
def vypis_prvocisel(kolik):  
    n = 1  
    while kolik > 0:  
        if len(seznam_delitelu(n)) == 2:  
            print n,  
            kolik -= 1  
    n += 1
```



## Test prvočíselnosti:

- zkusíme všechny možné dělitele od 2 do  $n - 1$
- vylepšení:
  - dělíme pouze dvojkou a lichými čísly
  - dělíme pouze dvojkou a čísly tvaru  $6k \pm 1$
  - dělíme pouze do  $\sqrt{n}$

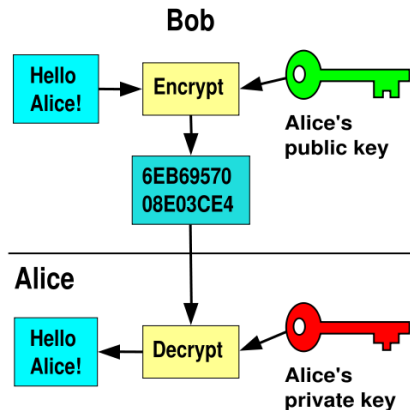
# Test prvočíselnosti: chytřejší algoritmy

- náhodnostní algoritmy
- polynomiální deterministický algoritmus (objeven 2002)
- (vysoce) nad rámec tohoto kurzu
- **umí se to** dělat rychle

# Rozklad na prvočísla

- rozklad na prvočísla = faktorizace
- naivní algoritmy:
  - průchod všech možných dělitelů
  - zlepšení podobně jako u testů prvočíselnosti
- chytřejší algoritmy:
  - složitá matematika
  - aktivní výzkumná oblast
  - **neumí se to** dělat rychle
  - max cca 200 ciferná čísla

# Příklad aplikace: asymetrická kryptologie



[http://en.wikipedia.org/wiki/Public-key\\_cryptography](http://en.wikipedia.org/wiki/Public-key_cryptography)

# Asymetrická kryptologie: realizace

- jednosměrné funkce
  - jednoduché vypočítat jedním směrem
  - obtížné druhým (inverze)
  - ilustrace: míchání barev
- RSA (Rivest, Shamir, Adleman) algoritmus
  - jednosměrná funkce: násobení prvočísel (inverze = faktorizace)
  - veřejný klíč: součin velkých prvočísel
  - bezpečnost  $\sim$  nikdo neumí provádět efektivně faktorizaci
  - využití modulární aritmetiky, Eulerovy věty, ...

# Eratosthenovo síto

- problém: výpis prvočísel od 2 do  $n$
- algoritmus: opakovaně provádíme
  - označ další neškrtnuté číslo na seznamu jako prvočíslo
  - všechny násobky tohoto čísla vyškrtni

# Eratosthenovo síto

1. krok

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60

2. krok

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
<del>21</del>	<del>22</del>	23	<del>24</del>	<del>25</del>	<del>26</del>	<del>27</del>	<del>28</del>	<del>29</del>	<del>30</del>	<del>31</del>	<del>32</del>	<del>33</del>	<del>34</del>	<del>35</del>	<del>36</del>	<del>37</del>	<del>38</del>	<del>39</del>	<del>40</del>
<del>41</del>	<del>42</del>	43	<del>44</del>	<del>45</del>	<del>46</del>	<del>47</del>	<del>48</del>	<del>49</del>	<del>50</del>	<del>51</del>	<del>52</del>	<del>53</del>	<del>54</del>	<del>55</del>	<del>56</del>	<del>57</del>	<del>58</del>	<del>59</del>	<del>60</del>

3. krok

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
<del>21</del>	<del>22</del>	23	<del>24</del>	<del>25</del>	<del>26</del>	<del>27</del>	<del>28</del>	<del>29</del>	<del>30</del>	<del>31</del>	<del>32</del>	<del>33</del>	<del>34</del>	<del>35</del>	<del>36</del>	<del>37</del>	<del>38</del>	<del>39</del>	<del>40</del>
<del>41</del>	<del>42</del>	43	<del>44</del>	<del>45</del>	<del>46</del>	<del>47</del>	<del>48</del>	<del>49</del>	<del>50</del>	<del>51</del>	<del>52</del>	<del>53</del>	<del>54</del>	<del>55</del>	<del>56</del>	<del>57</del>	<del>58</del>	<del>59</del>	<del>60</del>

4. krok

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
<del>21</del>	<del>22</del>	23	<del>24</del>	<del>25</del>	<del>26</del>	<del>27</del>	<del>28</del>	<del>29</del>	<del>30</del>	<del>31</del>	<del>32</del>	<del>33</del>	<del>34</del>	<del>35</del>	<del>36</del>	<del>37</del>	<del>38</del>	<del>39</del>	<del>40</del>
<del>41</del>	<del>42</del>	43	<del>44</del>	<del>45</del>	<del>46</del>	<del>47</del>	<del>48</del>	<del>49</del>	<del>50</del>	<del>51</del>	<del>52</del>	<del>53</del>	<del>54</del>	<del>55</del>	<del>56</del>	<del>57</del>	<del>58</del>	<del>59</del>	<del>60</del>

# Eratosthenovo síto

```
def eratosthenes(pocet):  
    je_kandidat = [ 1 for i in range(pocet) ]  
    for i in range(2, pocet):  
        if je_kandidat[i]:  
            print i,  
            k = 0  
            while k < pocet:  
                je_kandidat[k] = 0  
                k += i
```



# Funkcionální prvky v Pythonu

- funkcionální programování
  - výpočet jako vyhodnocení matematické funkce
  - předmět IB015, jazyk Haskell
- Python obsahuje funkcionální prvky, např.
  - generátorová notace seznamů (list comprehension)
  - funkce map, reduce, filter
  - lambda výrazy

## Funkcionální prvky v Pythonu – ukázka

```
n = 12
delitele = [ i for i in range(1, n+1) if n % i == 0 ]

print delitele
print map(str, delitele)
print map(lambda x: 'I'*x, delitele)
print filter(lambda x: x > 3, delitele)
print reduce(lambda x,y: x*y, delitele)

x = 3589
print sum(map(int, str(x))) # ciferny soucet
```

seznamy, řetězce:

- základní operace
- ukázky použití
- kryptografické příklady (historické) a souvislosti (moderní)
- příště: vyhledávání, řadicí algoritmy