Řetězce a seznamy (a kryptografické odbočky)

IB111 Úvod do programování

2016

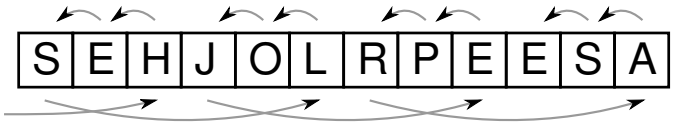
Rozcvička: šifry

1. C S A R B V
   E K T E O A
2. A J L B N O C E
3. C S B U J T M B W B

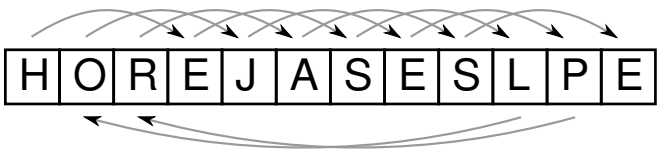
Transpoziční šifry

pozpátku

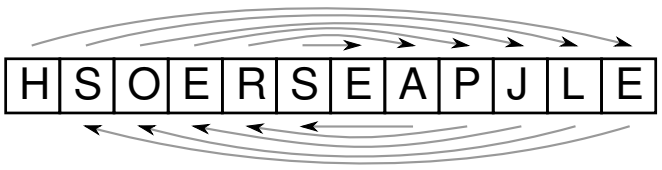
trojice pozpátku



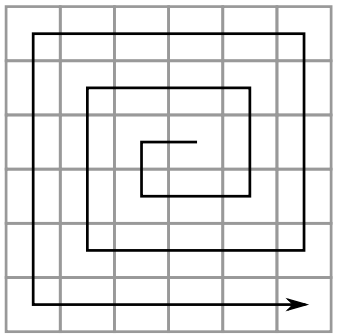
ob tři



dopředu dozadu

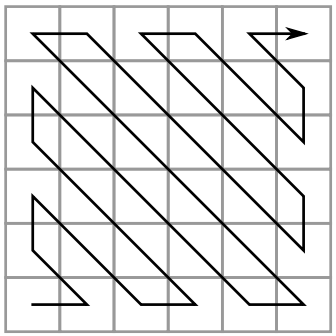


šnek



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
SLONSLONSLONSL
ZWSQWUDBVWWCGF
```

H → 7

S → 18

7 + 18 → 25 → 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
A	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

Indexování od nuly

Proč indexujeme od 0?

- částečně "historicky-technické" důvody
- ale i dobré "matematické" důvody

Pro zájemce:

`http://www.cs.utexas.edu/users/EWD/transcriptions/EWD08xx/EWD831.html`

`http://programmers.stackexchange.com/questions/110804/why-are-zero-based-arrays-the-norm`

`https://www.quora.com/Why-do-array-indexes-start-with-0-zero-in-many-programming-languages`

Kódování

- jak jsou znaky reprezentovány?
  ASCII, ISO 8859-2, Windows-1250, Unicode, UTF-8, ...
  `http://www.joelonsoftware.com/articles/Unicode.html`
  `http://www.polylab.dk/utf8-vs-unicode.html`
- Python3 – Unicode řetězce
- pro tento kurz:
  - `ord`, `chr` – převod znaků na čísla a zpět
  - anglická abeceda má přiřazena po sobě jdoucí čísla

```
for i in range(26):
    print(chr(ord('A')+i))
```

Řetězce – pokročilejší indexování

(specifické pro Python)

```
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 oproti 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 = "i Have a dream."
print(text.upper())
print(text.lower())
print(text.capitalize())
print(text.rjust(30))
print("X",text.center(30),"X")
print(text.replace("dream","nightmare"))
```

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

Pozn. objektová notace

Příklad: Transpozice (rozcvička 1)

- úkol: přepis textu po sloupcích
- příklad vstupu a výstupu (2 sloupce):
  - `CESKATREBOVA`
  - `C S A R B V`
    `E K T E O A`

Transpozice (rozcvička 1)

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

Transpozice (rozcvička 1), kratší varianta

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

Caesarova šifra (rozcvička 3)

- substituční šifra – posun v abecedě
- vstup: text, posun
- výstup: zašifrovaný text
- `BRATISLAVA`, $1 \rightarrow$ `CSBUJTMBWB`

Caesarova šifra – řešení

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

Pozn. Řešení má nedostatky – zkuste najít a vylepšit.

Caesarova šifra – rozlomení

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

Caesarova šifra – rozlomení

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

Caesarova šifra – rozlomení

k	Kandidát	b_s	b_f
0	MPKTWTDVLVELMZCF	0	21
1	NQLUXUEWMWFMNADG	13	0
2	ORMVYVFXNXGNOBEH	24	9
3	PSNWZWGYOYHOPCFI	5	-3
4	QTOXAXHZPZIPQDGJ	10	-6
5	RUPYBYIAQAJQREHK	0	9
6	SVQZCZJBRBKRSFIL	0	3
7	TWRADAKCSCLSTGJM	32	26
8	UXSBEBLDTDMTUHKN	0	24
9	VYTCFCMEUENUVILO	11	46
10	WZUDGDNFVFOVWJMP	0	-6
11	XAVEHEOGWGPWXKNQ	5	-2
12	YBWFIFPHXHQXYLOR	0	-28

k	Kandidát	b_s	b_f
13	ZCXGJGQIYIRYZMPS	0	-13
14	ADYHKHRJZJSZANQT	0	16
15	BEZILISKAKTABORU	**67**	**59**
16	CFAJMJTLBLUBCPSV	0	11
17	DGBKNKUMCMVCDQTW	5	-4
18	EHCLOLVNDNWDERUX	17	31
19	FIDMPMWOEOXEFSVY	5	22
20	GJENQNXPFPYFGTWZ	4	-23
21	HKFOROYQGQZGHUXA	16	-17
22	ILGPSPZRHRAHIVYB	28	18
23	JMHQTQASISBIJWZC	9	0
24	KNIRURBTJTCJKXAD	5	24
25	LOJSVSCUKUDKLYBE	4	29

Vigenèrova šifra

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

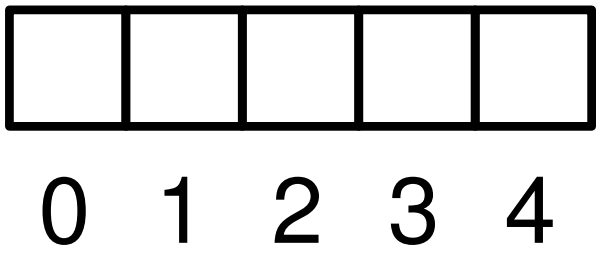
Seznamy (pole) – motivace

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

Frekvenční analýza nevhodně

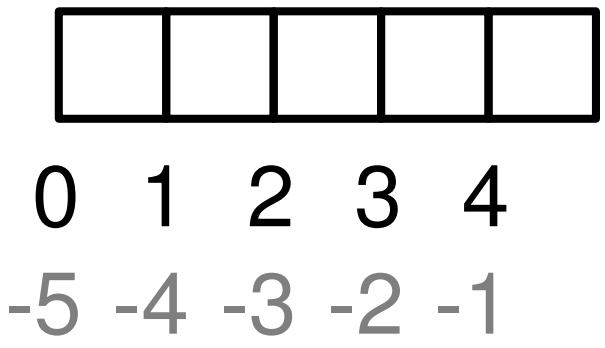
```
def frequency_analysis(text):
    text = text.upper()
    freqA = 0
    freqB = 0
    freqC = 0
    for letter in text:
        if letter == 'A':
            freqA += 1
        elif letter == 'B':
            freqB += 1
        elif letter == 'C':
            freqC += 1
    print('A', freqA)
    print('B', freqB)
    print('C', freqC)
```

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 (ale pomalejší)
- 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 pruku, s[2] = 1
s[-1]           # indexace od konce, s[-1] = 8
s[2] = 15       # změna pruku
s.append(6)     # přidání pruku
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

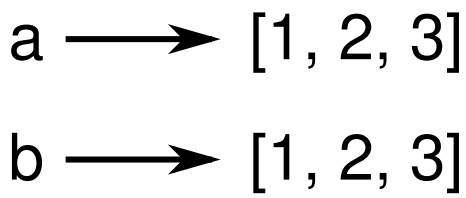
Python: seznamy a cyklus for

- cyklus `for` – přes prvky seznamu*
- `range` – vrací seznam* čísel
- typické použití: `for i in range(n)`
- ale můžeme třeba:
  - `for animal in ["dog", "cag", "pig"]: ...`
  - `for letter in "hello world": ...`

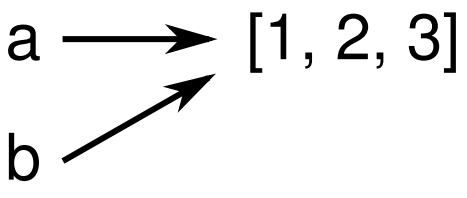
(*) ne úplně přesně – z důvodu efektivity se používají generátory a speciální „range object“, v případě potřeby použijte explicitní přetypování na seznam: `list(range(10))`

Objekty, hodnoty, aliasy – stručné varování

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



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



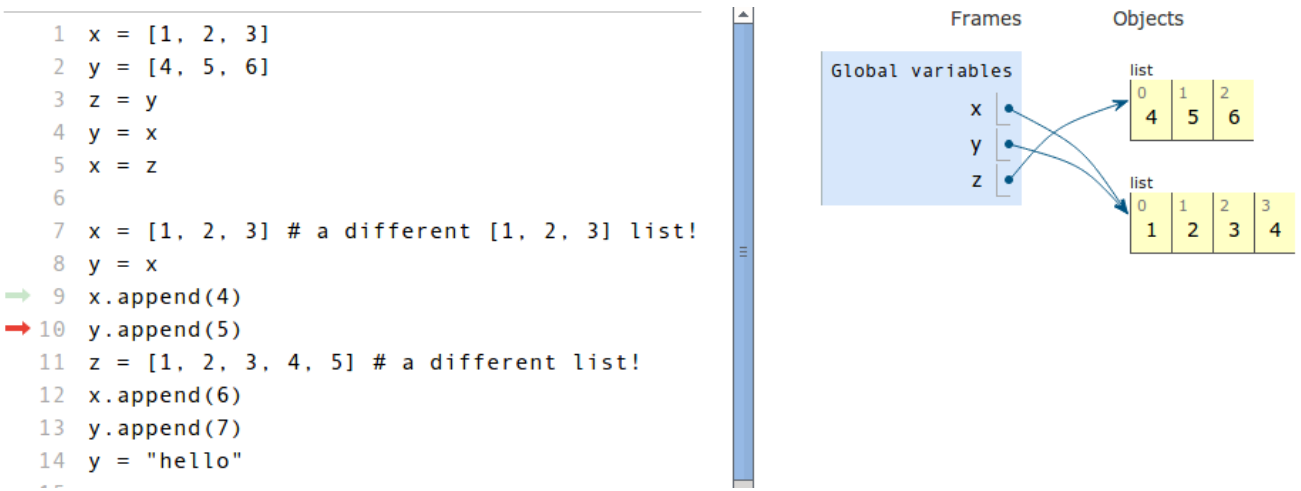
- parametry funkcí – pouze volání hodnotou
  (na rozdíl např. od Pascalu: volání hodnotou a odkazem)
- měnitelné objekty (např. seznam) však funkce může měnit
- mělká vs hluboká kopie
- více později

N-tice - stručné představení

- n-tice (tupples)
- neměnitelná varianta seznamů
- kulaté závorky místo hranatých:
  `t = (1, 2, 3)`
- neměnitelné podobně jako řetězce
- typické užití:
  - souřadnice: `(x,y)`
  - barva: `(r, g, b)`

Vizualizace běhu programu

`http://www.pythontutor.com/`



vhodné např. pokud je nejasný některý z příkladů ve slidech

Příklad: výpočet průměrné hodnoty

```
def average1(num_list):
    total = 0
    for i in range(len(num_list)):
        total += num_list[i]
    return total / len(num_list)

def average2(num_list):
    total = 0
    for x in num_list:
        total += x
    return total / len(num_list)

def average3(num_list):
    return sum(num_list) / len(num_list)
```

Ilustrace práce se seznamem

```
def divisors_list(n):
    divisors = []
    for i in range(1, n+1):
        if n % i == 0:
            divisors.append(i)
    return divisors

divisors24 = divisors_list(24)
print(divisors24)
print(len(divisors24))
for x in divisors24: print(x**2)
```

Frekvenční analýza nevhodně

```
def frequency_analysis(text):
    text = text.upper()
    freqA = 0
    freqB = 0
    freqC = 0
    for letter in text:
        if letter == 'A':
            freqA += 1
        elif letter == 'B':
            freqB += 1
        elif letter == 'C':
            freqC += 1
    print('A', freqA)
    print('B', freqB)
    print('C', freqC)
```

Frekvenční analýza lépe

```
def frequency_analysis(text):
    text = text.upper()
    frequency = [ 0 for i in range(26) ]
    for letter in text:
        if ord(letter) >= ord('A') and\
           ord(letter) <= ord('Z'):
            frequency[ord(letter) - ord('A')] += 1
    for i in range(26):
        if frequency[i] != 0:
            print(chr(ord('A')+i), frequency[i])
```

Simulace volebního průzkumu – nevhodné řešení

```
def survey(size, pref1, pref2, pref3):
    count1 = 0
    count2 = 0
    count3 = 0
    for i in range(size):
        r = random.randint(1,100)
        if r <= pref1: count1 += 1
        elif r <= pref1 + pref2: count2 += 1
        elif r <= pref1 + pref2 + pref3: count3 += 1
    print("Party 1:", 100.0 * count1 / size)
    print("Party 2:", 100.0 * count2 / size)
    print("Party 3:", 100.0 * count3 / size)
```

Simulace volebního průzkumu – lepší řešení

```
def survey(size, pref):
    n = len(pref)
    count = [ 0 for i in range(n) ]
    for _ in range(size):
        r = random.randint(1,100)
        for i in range(n):
            if sum(pref[:i]) < r <= sum(pref[:i+1]):
                count[i] += 1
    for i in range(n):
        print("Party", i+1, 100 * count[i] / size)
```

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 to_morse_poor(text):
    result = ''
    for i in range(len(text)):
        if text[i] == 'A': result += '.-|'
        elif text[i] == 'B': result += '-...|'
        elif text[i] == 'C': result += '-.-.|'
        elif text[i] == 'D': result += '-..|'
        # etc
    return result
```

Převod do Morseovy abecedy: využití seznamu

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

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

(ještě lepší řešení: využití slovníku – bude později)

Převod z Morseovy abecedy

```
def find_letter(sequence):
    for i in range(len(morse)):
        if morse[i] == sequence:
            return chr(ord('A') + i)
    return '?'

def from_morse(message):
    result = ''
    sequence = ''
    for symbol in message:
        if symbol == '|':
            result += find_letter(sequence)
            sequence = ''
        else:
            sequence += symbol
    return result
```

Split – seznam z řetězce

`split` – rozdělí řetězec podle zadaného oddělovače, vrátí seznam

```
>>> vowels = "a,e,i,o,u,y"
>>> vowels.split(",")
['a', 'e', 'i', 'o', 'u', 'y']
>>> message = ".-..|---|-..|.-"
>>> message.split("|")
['.-..', '---', '-..', '.-']
```

Příklad – načítání vstupu od uživatele

```
>>> input_string = input()
3 7
>>> xstring, ystring = input_string.split(" ")
>>> x = int(xstring)
>>> y = int(ystring)
```

Výškový profil

mapy.cz

Výškový profil

```
heights_profile([3,4,5,3,4,3,2,4,5,6,5])
```

```
                  #
    #           # # #
  # #   #     # # # #
# # # # # #   # # # #
# # # # # # # # # # #
# # # # # # # # # # #
```

```
Ascent 7
Descent 5
```

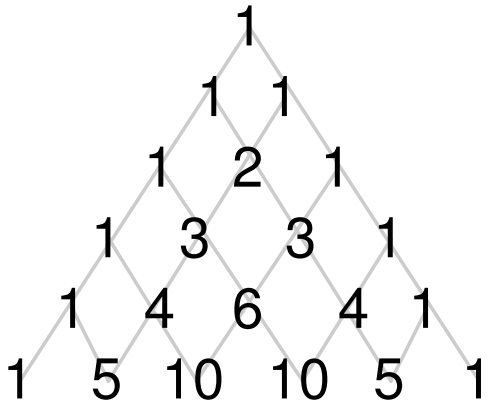
Výškový profil

```
def heights_profile(heights):
    for v in range(max(heights)):
        for i in range(len(heights)):
            if heights[i] >= max(heights) - v:
                print("#", end=" ")
            else:
                print(" ", end=" ")
        print()
    print()
```

Výškový profil

```
def elevation(heights):
    ascent = 0
    descent = 0
    for i in range(len(heights)-1):
        if heights[i] < heights[i+1]:
            ascent += heights[i+1] - heights[i]
        else:
            descent += heights[i] - heights[i+1]
    print("Ascent", ascent)
    print("Descent", descent)
```

Pascalův trojúhelník



$$\binom{0}{0}$$
$$\binom{1}{0}\ \binom{1}{1}$$
$$\binom{2}{0}\ \binom{2}{1}\ \binom{2}{2}$$
$$\binom{3}{0}\ \binom{3}{1}\ \binom{3}{2}\ \binom{3}{3}$$

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 pascal_triangle(n):
    current_row = [ 1 ]
    for j in range(n):
        for x in current_row:
            print(x, end=" ")
        print()
        next_row = [ 1 ]
        for i in range(len(current_row)-1):
            next_row.append(current_row[i] +\
                            current_row[i+1])
        next_row.append(1)
        current_row = next_row
```

Prvočísla

- 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 print_primes(how_many):
    n = 1
    while how_many > 0:
        if len(divisors_list(n)) == 2:
            print(n, end=" ")
            how_many -= 1
        n += 1
    print()
```

Odbočka: test prvočíselnosti, kryptografie

Test prvočíselnosti:

- zkoušíme 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 $\sqrt{n}$

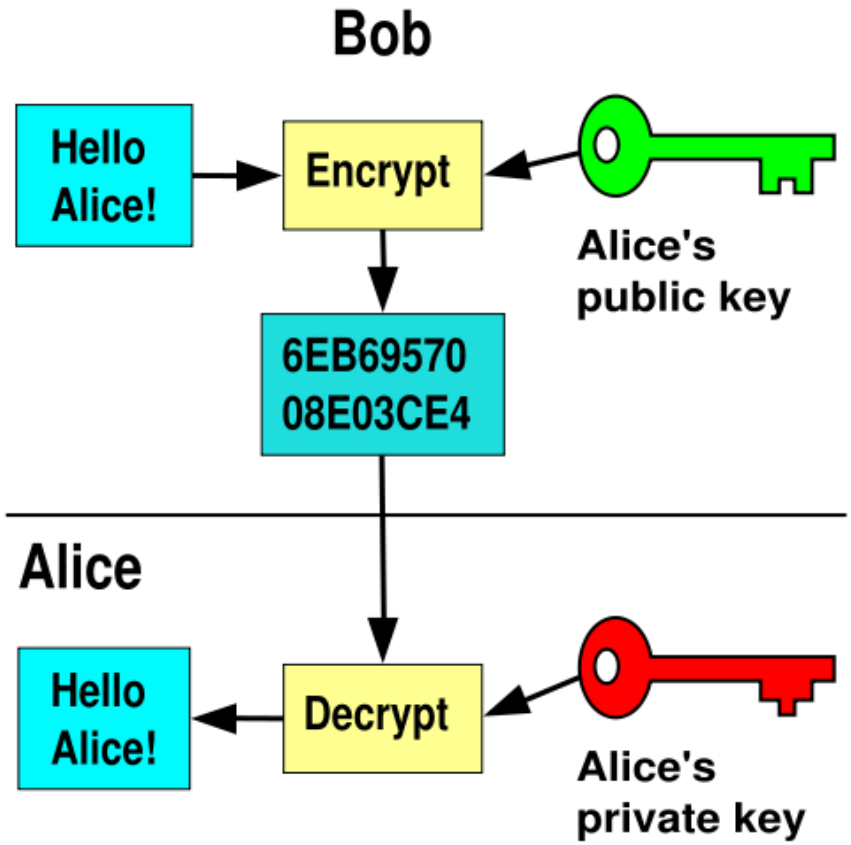
Test prvočíselnosti: chytřejší algoritmy

- 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

Asymetrická kryptologie: realizace

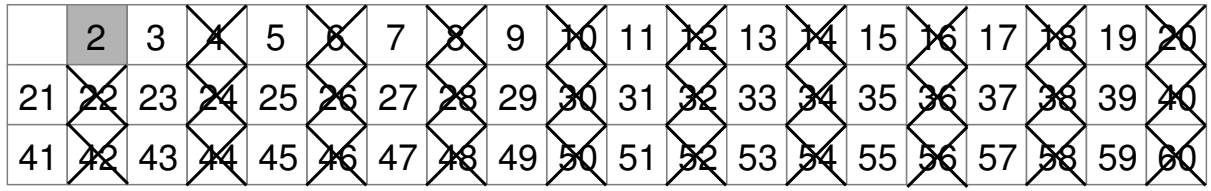
- jednosměrné funkce
  - jednoduché vypočítat jedním směrem
  - obtížné druhým (inverze)
  - ilustrace: míchání barev
- RSA (Rivest, Shamir, Adleman) algoritmus
  - jednosměrná funkce: násobení prvočísel (inverze = faktorizace)
  - veřejný klíč: součin velkých prvočísel
  - bezpečnost $\sim$ nikdo neumí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

Eratosthenovo síto

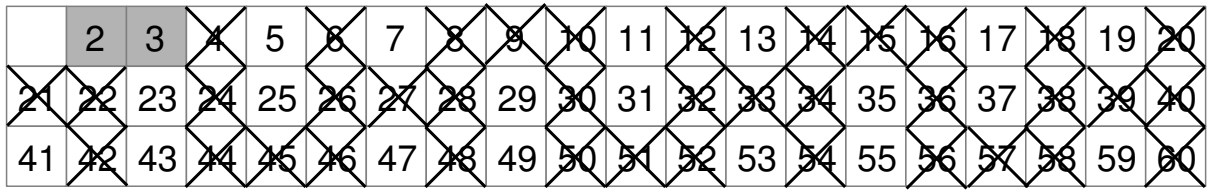
- problém: výpis prvočísel od 2 do n
- algoritmus: opakovaně provádíme
  - označ další neškrtnuté číslo na seznamu jako prvočíslo
  - všechny násobky tohoto čísla vyškrtni

Eratosthenovo síto

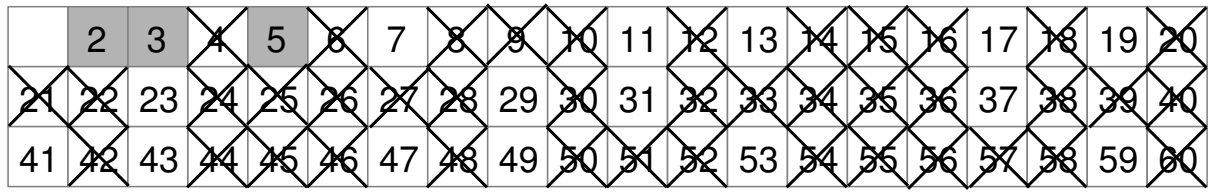
1. krok



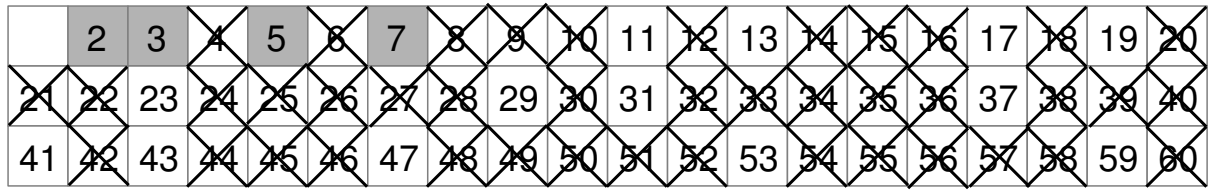
2. krok



3. krok



4. krok



Eratosthenovo síto

```
def eratosthenes(count):
    is_candidate = [ 1 for i in range(count) ]
    for i in range(2, count):
        if is_candidate[i]:
            print(i, end=" ")
            k = 0
            while k < count:
                is_candidate[k] = 0
                k += i
    print()
```

Zajímavosti

- prvočísla – Ulamova spirála
- Pascalův trojúhelník – obarvení podle sudosti – Sierpiského trojúhelník

Vi Hart: Doodling in math: Sick number games

`https://www.khanacademy.org/math/recreational-math/vi-hart/doodling-in-math/v/doodling-in-math-sick-number-games`

Praktické rady: Chyby, ladění

- čtení chybových hlášek
- časté chyby

Čtení chybových hlášek

```
Traceback (most recent call last):
  File "vyhledavani-razeni-demo.py", line 63, in <module>
    test_sorts()
  File "vyhledavani-razeni-demo.py", line 59, in test_sorts
    sort(a)
  File "vyhledavani-razeni-demo.py", line 52, in insert_sort
    a[j] = curent
NameError: name 'curent' is not defined
```

- kde je problém? (identifikace funkce, číslo řádku)
- co je za problém (typ chyby)

Základní typy chyb

- SyntaxError
  - invalid syntax: zapomenutá dvojtečka či závorka, záměna = a ==, ...
  - EOL while scanning string literal: zapomenutá uvozovka
- NameError – špatné jméno proměnné (překlep v názvu, chybějící inicializace)
- IndentationError – špatné odsazení
- TypeError – nepovolená operace (sčítání čísla a řetězce, přiřazení do řetězce, ...)
- IndexError – chyba při indexování řetězce, seznamu a podobně („out of range“)

Časté chyby

projeví se „rychle“ (program spadne hned):

- zapomenutá dvojtečka, závorka, uvozovka
- překlepy
- použití $=$ tam, kde mělo být $==$
- špatný počet argumentů při volání funkce
- zapomenuté `len` ve `for i in range(alist)`

Časté chyby

nemusí se projevit rychle / vždy:

- použití == tam, kde mělo být =
- `"True"` místo `True`
- chybné indexování (řetězce, seznamy)
- záměna `print` a `return`
- dělení nulou

Shrnutí

Seznamy, řetězce:

- základní operace
- ukázky použití
- kryptografické příklady (historické) a souvislosti (moderní)

Příště: Vyhledávání, řadicí algoritmy