Algoritmy a programování

Řetězce, práce se soubory

Vojtěch Vonásek

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Řetězce (string)

- Složený datový typ
- Lze vytvořit jednoduchými nebo dvojitými uvozovkami

```
1 a = "ahoj"
2 b = 'hello'
```

- Oba způsoby jsou ekvivalentní
- Datový typ string je immutable, tj. nelze měnit jeho vnitřní hodnoty
- Délka řetězce: funkce `len()`
  - `print(len("ahoj"))` $\rightarrow$ `4`
  - `print(len("0"))` $\rightarrow$ `1`
  - `print(len(""))` $\rightarrow$ `0`
- Přístup na jednotlivá písmena — operátor `[]`
  - `a[i]` je i-tý znak v proměnné `a` (číslujeme od 0)
  - `a[0]` je první znak v proměnné `a`
  - `a[len(a)-1]` je poslední znak

Řetězce: vytváření

- Inicializace: operátor =

```
1 emptyString = ""
2 otherString = "some␣text"
```

- Spojování: operátor +

```
1 intro = "Hello"
2 name = "John"
3 text = intro + ",␣" + name
```

- Opakování: operátor *

```
1 fiveSpaces = "␣"*5
```

- Spojení (ne-stringových) proměnných: přetypování funkcí `str()`

```
1 number = 10/2
2 text = "The␣number␣is␣" + str(number)
```

Řetězce: indexování

- Přístup na jednotlivá písmena — operátor `[]`
- Nelze přistupovat na neexistující prvek!

```
1 a = "ahoj"
2 print(a[10])
```

```
    print(a[10])
IndexError: string index out of range
```

Řetězce: použití for+range

- Přístup na jednotlivá písmena — operátor `[]`
- V řídicí proměnné je index

```
1 a = "ahoj"
2 for i in range(len(a)):
3     print(a[i])
```

```
a
h
o
j
```

Řetězce: použití for+in

- For cyklus s operátorem `in` použitý na proměnnou typu string vrací písmena zleva doprava

```
1 a = "ahoj"
2 for letter in a:
3     print(letter)
```

```
a
h
o
j
```

Řetězce: procházení přes for cyklus a []

- Pozor, string je immutable, tj. nelze ho měnit!

```
1 a = "pepa"
2 a[0] = "P"
```

```
    a[0] = "P"
TypeError: 'str' object does not support item
   assignment
```

```
1 a = "ahoj"
2 for i in range(len(a)):
3     a[i] = "*"
```

```
    a[i] = "*"
TypeError: 'str' object does not support item
   assignment
```

Řetězce: použití for+in

- Pozor, string je immutable, tj. nelze ho měnit!

```
1 a = "abc"
2 print(a)
3 for i in a:
4     i = "*"
5     print(i)
6 print(a)
```

```
abc
*
*
*
abc
```

- Zde měníme hodnotu proměnné `i` (přepíšeme na `'*'`)
- Ale neměníme konkrétní prvek v proměnné `a`

Řetězce: úpravy

- Stringy jsou immutable, nelze je měnit přímo přes opeátor `[]`
- Úpravy (a další operace) s využitím metod třídy String
- Náhrada podřetězce: metoda `replace(oldString, newString)`

```
1 x = "Hahaha!"
2 y = x.replace("a","e")  #x is not changed
3 print(x)
4 print(y)
```

```
Hahaha!
Hehehe!
```

```
1 x = "Python␣is␣cool"
2 y = x.replace("Python","c++")  #x is not changed
3 print(x)
4 print(y)
```

```
Python is cool
c++ is cool
```

Řetězce: úpravy

- Stringy jsou immutable, nelze je měnit přímo přes opeátor `[]`
- Úpravy (a další operace) s využitím metod třídy String
- Náhrada části řetězce: metoda `replace(oldString, newString)`

```
1 x = "text␣to␣remove␣text"
2 y = x.replace("text","")
3 print(x)
4 print(y)
```

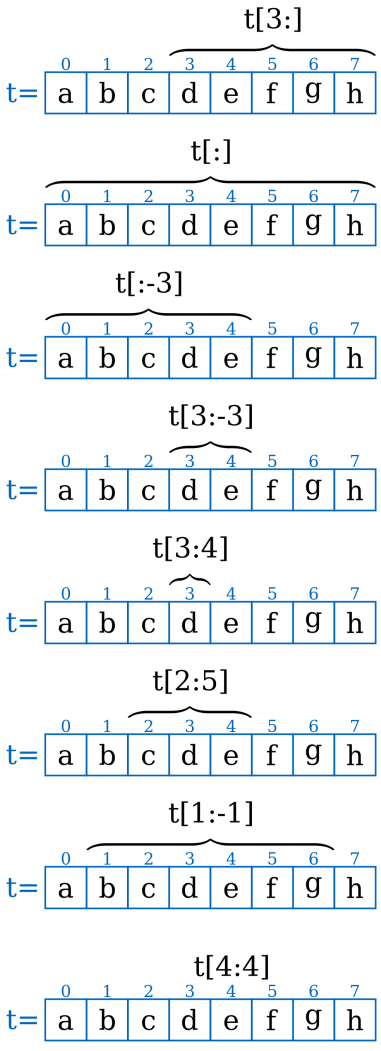
```
text to remove text
 to remove
```

Řezy (slices)

- `t[i:j]` vrací řetězec od pozice i do pozice j **(kromě j)**
- `t[i:]` od pozice i do konce
- `t[:j]` od začátku do pozice j **(kromě j)**
- `t[:]` kopie celého řetězce
- Výsledek řezu řetězce je opět řetězec

```
1  t = "abcdefgh"
2  print(t[3:])
3  print(t[:])
4  print(t[:-3])
5  print(t[3:-3])
6  print(t[3:4])
7  print(t[2:5])
8  print(t[1:-1])
9  print(t[4:4])   #!
10 print("*")
```

```
defgh
abcdefgh
abcde
de
d
cde
bcdefg

*
```



Výpis na obrazovku

`print()`

- Přidává automaticky konec řádku (`\n`)
- Pokud má více argumentů, vkládá mezi ně mezeru

```
1 print("a","b","c")
2 print("*")
```

```
a b c
*
```

- Další argumenty funkce `print()`
  - `sep` — oddělovač, znak mezi argumenty (defaultně mezera)
  - `end` — znak, který se tiskne na konci výpisu (defaultně `\n`)
- `print(a,b,c)` je ekvivaletní `print(a,b,c,sep=" ",end="\n")`

print(): ovlivnění výstupu

- Změnou `sep` nebo `end` můžeme ovlivnit způsob výpisu

```
1 a=1
2 b=2
3 print(a,b, end="|")
4 print("*")
```

```
1 2|*
```

```
1 a = 1
2 b = 2
3 c = 3
4 print(a,b,c, end="\n\n", sep="@")
5 print("*")
```

```
1@2@3


*
```

Zápis řetězců, speciální znaky

- Vytvoření řetězců z písmen abecedy a čísel je triviální
- Jak zapíšeme jiné znaky?

Doplňte program:

```
1 a =
2 print(a)
```

aby vypsal:

```
Peter's son is "Paul"
```

ASCII tabulka

- Datové typy jsou v počítači uloženy v bytech
- V případě řetězců (a txt souborů) se data “zobrazují” ve formě znaků
- Pro správnou interpretaci bytů je třeba znát konverzní tabulku
- ASCII a UTF-8

ASCII

- Definuje význam pro 7bitů (0–127)
- Řídicí znaky (0–31), též netisknutelné znaky
  - Ovlivňují výstup, např. nový řádek ($\backslash n$), tabulátor ($\backslash t$)
- Tisknutelné znaky (32–127)
  - Znaky angl. abecedy, čísla, mezera, závorky, znamínka . . .
- Extended ASCII (8 bitů, rozsah 128-255)
  - Znaky národních abeced
  - Pro každý jazyk je jiné rozšíření

ASCII tabulka

ASCII control characters

00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)
18	DC2	(Device control 2)
19	DC3	(Device control 3)
20	DC4	(Device control 4)
21	NAK	(Negative acknowl.)
22	SYN	(Synchronous idle)
23	ETB	(End of trans. block)
24	CAN	(Cancel)
25	EM	(End of medium)
26	SUB	(Substitute)
27	ESC	(Escape)
28	FS	(File separator)
29	GS	(Group separator)
30	RS	(Record separator)
31	US	(Unit separator)
127	DEL	(Delete)

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Extended ASCII characters

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

ASCII tabulka

- `ord(letter)`: vrátí číslo znaku v ASCII tabulce
- `chr(number)`: vrátí znak dle ASCII

```
1 text = "ahoj"
2 for letter in text:
3     print("Letter",letter,", ord:", ord(letter))
4
5 compose = chr(65) + chr(32) + chr(64)
6 print(compose)
```

```
Letter a , ord: 97
Letter h , ord: 104
Letter o , ord: 111
Letter j , ord: 106
A @
```

UTF-8

- Kódování s proměnlivou délkou (znak je kódován 1 až 4 byty)
- Lze reprezentovat až 1,112,064 znaků (včetně znaků národních abeced)
- Význam prvních 7 bitů (0–127) je shodný s ASCII
- Python nativně podporuje UTF-8
- Python předpokládá, že zdrojový program je vytvořený v UTF-8
- Vložení UTF-8 znaků:`\uCODE`, kde CODE je číslo znaku v UTF-8

```
1 print("k\u016f\u0148")
```

```
kůň
```

Escape sekvence

- Escape sekvence: zápis vybraných řídicích znaků

\'	apostrof
\"	uvozovky
\\	zpětné lomítko
\n	nový řádek (new line, line feed)
\r	nový řádek (carriage return)
\t	tabulátor
\b	backspace

```
1 a = "first␣line\nsecond␣line\nthird␣line\n"
2 print(a, end="|")
3 print("normal␣print")
4 b = "\t\tindented␣text\n";
5 print(b,end="|")
```

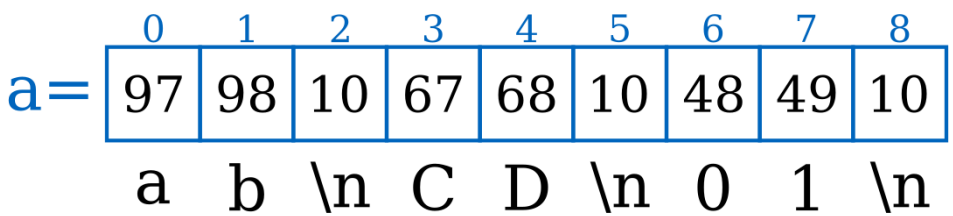
```
first line
second line
third line
|normal print
                    indented text
|
```

Reprezentace řetězce

- Znaky jsou v počítači uložené v bytech
- Řetězec je sekvence znaků, každý znak je 8-mi bitové číslo (byte)
- Zobrazení řetězců (a txt souborů) přes konverzní tabulku (typicky ASCII, UTF-8)

```
1 a = "ab\nCD\n01\n";
2 print(a)
3 for letter in a:
4     print(ord(letter), end="␣")
```

```
ab
CD
01

97 98 10 67 68 10 48 49 10
```



- Znalost použitého kódování je důležitá pro správnou interpretaci (editorem, titulky, browser . . .)
- Ale i pro správnou interpretaci Pythonem
- Python předpokládá, že vstup (program + vstupní soubory) je UTF-8
- Pokud ne, dojde k nesprávné interpretaci znaků
- To může vést na chybu programu, nekonečný cyklus . . .

```
Mě to běží správně, ale Brute mi nepřiděl body!
MĚ~[ to bĚ~[ĹžĚ  sprĚĄvnĚ~[, ale Brute mi nepĹ~YidĚ~[lil body!
MΔ~[ to bΔ~[EYΓ  sprΓ‘vnΔ~[, ale Brute mi nepE~YidΔ~[lil body!
```

Coding

- Znalost použitého kódování je důležitá pro správnou interpretaci (editorem/Pythonem)
- Python předpokládá, že vstup je UTF-8
- Pokud ne, dojde k nesprávnému pochopení znaků (chyba programu, nekonečný cyklus apod)
- UTF-8

```
2 číslo = 1
3 dalšíČíslo = číslo + 2
4 print(číslo, dalšíČíslo)
```

```
1  3
```

- Jiné kódování (8859-2)

```
2 èíslo = 1
3 dal¹íÈíslo = èíslo + 2
4 print(èíslo, dal¹íÈíslo)
```

```
File "program8859-2.py", line 1
SyntaxError: Non-UTF-8 code starting with '\xe8' in file program8859-2.py
 on line 1, but no encoding declared; see http://python.org/dev/peps/pep-
0263/ for details
```

Doporučení

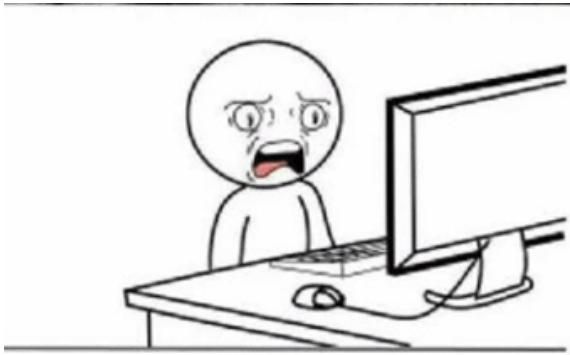
- V programech nepoužíváme české (a jiné národní) znaky
- Ani v komentářích

```
2 èíslo = 1
3 dal¹íÈíslo = èíslo + 2
4 print(èíslo, dal¹íÈíslo)
```

```
File "program8859-2.py", line 1
SyntaxError: Non-UTF-8 code starting with '\xe8' in file program8859-2.py
 on line 1, but no encoding declared; see http://python.org/dev/peps/pep-
0263/ for details
```

```
1   class 💩💩💩💩 {
2       func 💩💩💩(😎: Int, 🐶: Int) -> Int {
3           return 😎 + 🐶;
4       }
5   }
6
7   let 🐔 = 3;
8   let 😢 = 🐔 + 2;
9   let 💩 = 💩💩💩💩();
10  print(💩.💩💩💩(🐔, 🐶:😢));
11
```

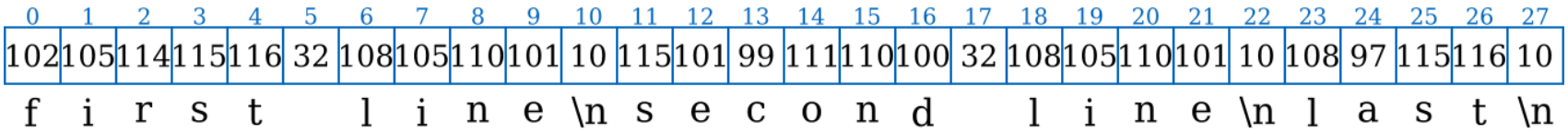
- Toto v Pythonu nelze, proměnná musí obsahovat pouze alfanumerické znaky případně podtržítko



```
namespace 🔵 = std;
using 🔢 = int;
using 💀 = void;
using 🕐 = time_t;
using 👌 = bool;
#define 👂 auto
#define 👀 enum
#define 👎 false
#define 👍 true
#define 😈 "evil"
#define 💪 🔵::make_shared
#define 🍸 virtual
#define 🖥️ 🔵::cout
#define 🚽 🔵::endl
template<class 🔮>
using 📚 = 🔵::vector<🔮>;
template<class 🔮>
using 👇 = 🔵::shared_ptr<🔮>;

👀 🐕 { 🐶, 🐕, 🐩, 🐺 };
🔢 🎲() { return 🔵::rand(); }
👌 😎() { return 👎; }

struct 🍴 { 🍸 💀 👓() = 0; };
struct 🍊 : 🍴 { 🍸 💀 👓() { 🖥️ << "🍊" << 🚽; }; };
struct 🍉 : 🍴 { 🍸 💀 👓() { 🖥️ << "🍉" << 🚽; }; };
struct 🍒 : 🍴 { 🍸 💀 👓() { 🖥️ << "🍉" << 🚽; }; };
struct 🍓 : 🍴 { 🍸 💀 👓() { 🖥️ << "🍓" << 🚽; }; };
struct 🍈 : 🍴 { 🍸 💀 👓() { 🖥️ << "🍈" << 🚽; }; };
struct 🍎 : 🍴 { 🍸 💀 👓() { 🖥️ << "🍎" << 🚽; }; };

🔢 main()
{
    if (😎() == 👎)
        🖥️ << "💩" << 🚽;

    📚<👇<🍴>> 🍽️ = { 💪<🍊>(), 💪<🍉>(), 💪<🍒>(), 💪<🍈>(), 💪<🍎>() };

    for (👂 🍏 : 🍽️)
        🍏->👓();

    return 🎲();
}
```

Soubory: reprezentace

- Soubor je posloupnost bytů

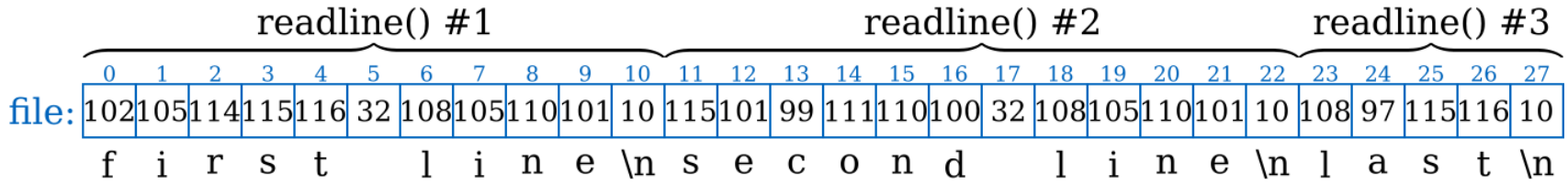


- Pokud je soubor považován za textový, jsou jednotlivé byty (nebo jejich skupiny) interpretovány jako znaky (dle konverzní tabulky)
- Často používané tabulky: ASCII, UTF-8
- V Pythonu se defaultně používá UTF-8
- Zobrazení souboru bude následující

```
first line
second line
last
```

Soubory: čtení

- Otevření pro čtení: `f = open(jmeno_souboru, "rt")`
- ”rt”— ”read text”: soubor je otevřen pro **čtení textového** souboru
- Po otevření ukazuje `f` (“handle”) na začátek souboru
- `readline()`: string znaků od aktuální pozice do prvního konce řádku




```
1 f = open("simple1.txt", "rt")
2 print('Line:',f.readline(),sep="",end="|") #1
3 print('Line:',f.readline(),sep="",end="|") #2
4 print('Line:',f.readline(),sep="",end="|") #3
5 print('Line:',f.readline(),sep="",end="|") #4
6 f.close()
```

```
Line:first line
|Line:second line
|Line:last
|Line:|
```

simple1.txt

```
first line
second line
last
```

Soubory: čtení

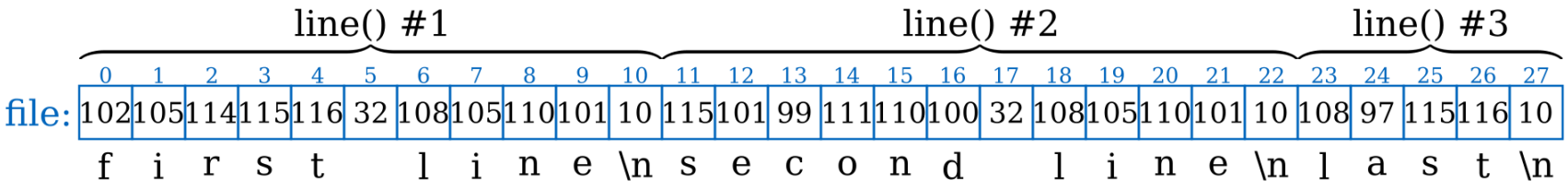
- Načtení všech řádků for cyklem
- For cyklus automaticky skončí po načtení posledního řádku

```
1 f = open("simple1.txt", "rt")
2 for line in f:
3     print("Line:", line, end="|")
4 f.close()
```

```
Line: first line
|Line: second line
|Line: last
|
```

simple1.txt

```
first line
second line
last
```



- Součástí načteného řetězce je konec řádku `'\n'`

Soubory: čtení

- Načtení všech řádků for cyklem
- For cyklus automaticky skončí po načtení posledního řádku
- Hledání řetězce v souboru

```
1 fread = open("names.txt", "rt")
2 toBeFound = "Diesel"
3 numberFound = 0
4 for line in fread:
5     if line == toBeFound:
6         numberFound+=1
7 fread.close()
8 print(toBeFound,"is", numberFound, "x in the file")
```

```
Diesel is 0 x in the file
```

names.txt

```
Minnie
Diesel
Zara
Grace
Rudy
Diesel
Cocoa
Gucci
Belle
Diesel
Allie
Harley
Tiger
Mickey
```

- Proč program nefunguje?

Soubory: čtení

- Načtení všech řádků for cyklem
- For cyklus automaticky skončí po načtení posledního řádku
- Hledání řetězce v souboru

```
1 fread = open("names.txt", "rt")
2 toBeFound = "Diesel"
3 numberFound = 0
4 for line in fread:
5     line = line.strip()
6     if line == toBeFound:
7         numberFound+=1
8 fread.close()
9 print(toBeFound,"is", numberFound, "x␣in␣the␣file")
```

```
Diesel is 3 x in the file
```

names.txt

```
Minnie
Diesel
Zara
Grace
Rudy
Diesel
Cocoa
Gucci
Belle
Diesel
Allie
Harley
Tiger
Mickey
```

- Součástí načteného stringu je konec řádku `'\n'`
- Použijeme `line.strip()` pro jeho odstranění

Soubory: čtení

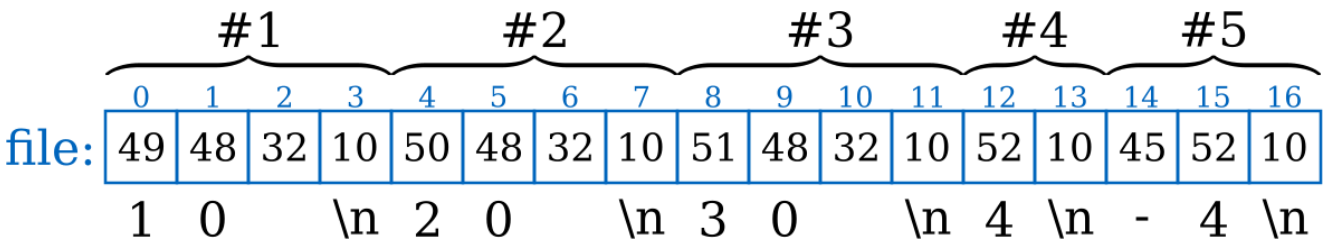
- Načtení všech řádků for cyklem
- For cyklus automaticky skončí po načtení posledního řádku
- Uložení dat do pole

```
1 f = open("simple2.txt", "rt")
2 numbers = []
3 for line in f:
4     a = int(line.strip())
5     numbers.append(a)
6 f.close()
7 print(numbers)
```

simple2.txt

```
10
20
30
4
-4
```

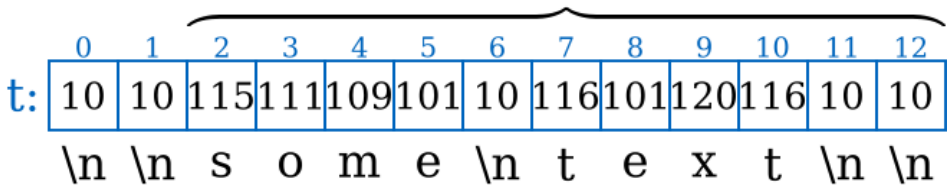
```
[10, 20, 30, 4, -4]
```



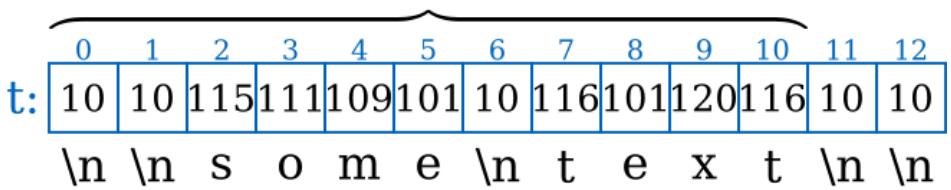
Soubory: čtení

- Konce řádků `'\n'` jsou součástí načtených řetězců
- `t.lstrip()`: vrací string, kde jsou zleva odstraněny formátovací znaky
- `t.rstrip()`: vrací string, kde jsou zprava odstraněny formátovací znaky
- `t.strip()`: kombinace `lstrip()` a `rstrip()`

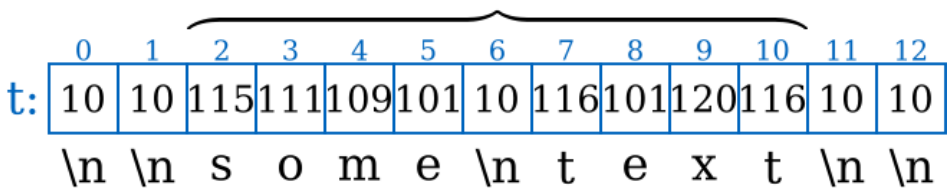
t.lstrip()='some\ntext\n\n'



t.rstrip()='\n\nsome\ntext'



t.strip()='some\ntext'



Soubory: zápis


```
1 f = open(jmeno_souboru, "wt")
```

- "wt"— "write text": soubor je otevřen pro **zápis textového** souboru
- Pokud soubor existuje, je přepsán, jinak je vytvořen
- Do souboru zapisujeme řetězce: `f.write(text)`

```
1 f = open("someFile.txt","wt")
2 f.write("ABC")
3 f.write("def\n")
4 b = 123
5 f.write(str(b) + "␣" + str(1/7) )
6 f.close()
```

someFile.txt

```
ABCdef
123 0.14285714285714285
```

Soubory: `close()`

- Po skončení práce se souborem je dobré zavolat `close()`
- Python sice zavře všechny otevřené soubory při skončení programu, ale není to efektivní
- Počet otevřených souborů je pro každý proces omezen (1024 na Ubuntu/Linux)
- Při neuzavírání souborů se zbytečně čerpají systémové prostředky
- Nevolání `close()` může vést na chyby

```
1 f = open("someFile.txt","wt")
2 f.write("ABC")
3 f.write("def\n")
4 b = 123
5 f.write(str(b) + "␣" + str(1/7) )
6 f.close()
```

Soubory: `close()`

- Vygenerujeme náhodný počet hvězdiček, uložíme je do souboru a znovu načteme
- Porovnáme uložený řetězec a řetězec načtený ze souboru

```
1  import random
2  for i in range(8):
3      n = random.randint(1,20)
4      stars = "*" * n
5      fo = open("tempfile.txt","wt")
6      fo.write(stars)
7
8      fi = open("tempfile.txt","rt")
9      word = fi.readline().strip()
10     print("Saved:", stars, len(stars),", loaded:", word,len(word))
```

```
Saved: ************** 14 , loaded:  0
Saved: ***************** 16 , loaded: ************** 14
Saved: ***************** 15 , loaded: ***************** 16
Saved: ********** 10 , loaded: *************** 15
Saved: ***************** 16 , loaded: ********** 10
Saved: ******************* 18 , loaded: ***************** 16
Saved: ************ 13 , loaded: ******************* 18
Saved: ***************** 16 , loaded: ************ 13
```

Soubory: `close()`

- Používejte `close()`

```
1  import random
2  for i in range(8):
3      n = random.randint(1,20)
4      stars = "*" * n
5      fo = open("tempfile.txt","wt")
6      fo.write(stars)
7      fo.close()
8
9      fi = open("tempfile.txt","rt")
10     word = fi.readline().strip()
11     print("Saved:", stars, len(stars),", loaded:", word,len(word))
12     fi.close()
```

```
Saved: ****************** 18 , loaded: ****************** 18
Saved: ******************** 20 , loaded: ******************** 20
Saved: ***** 5 , loaded: ***** 5
Saved: ******* 7 , loaded: ******* 7
Saved: ****************** 18 , loaded: ****************** 18
Saved: *************** 15 , loaded: *************** 15
Saved: ***************** 17 , loaded: ***************** 17
Saved: ****************** 18 , loaded: ****************** 18
```

Soubory: doporučení

- Uvádět explicitně, jestli má být soubor otevřen pro čtení (`"rt"`) nebo zápis (`"wt"`)

```
fread = open(jmeno_souboru, "rt")
fwrite = open(jmeno_souboru, "wt")
```

- Pomáhá čtení (pochopení) programu, pomáhá při hledání chyb
- Načtení dat
  - `for line in fread: ...`
  - `line = fread.readline()` (je třeba volat opakovaně pro každý řádek)
- Odstraníme konce řádků: `line.strip()`
- Po skončení práce se souborem ho **vždy zavřeme**: `fread.close()` nebo `fwrite.close()`

Soubory: příklad

Textový soubor obsahuje postupně:

- Jeden řádek s počtem slov *n* ve skupině 1
- Jeden řádek s počtem slov *m* ve skupině 2
- *n* řádků (každý obsahuje jedno slovo)
- *m* řádků (každý obsahuje jedno slovo)

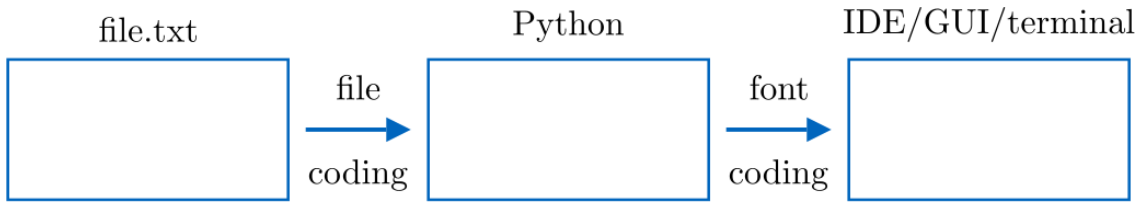
```
2
4
temple
exit
realm
nave
desire
thrust
```

```
1 fread = open("groups.txt", "rt")
2 size1 = int( fread.readline().strip() )
3 size2 = int( fread.readline().strip() )
4 for i in range(size1):
5     print("Group1:",fread.readline().strip())
6 for i in range(size2):
7     print("Group2:",fread.readline().strip())
8 fread.close()
```

```
Group1: temple
Group1: exit
Group2: realm
Group2: nave
Group2: desire
Group2: thrust
```

Coding podrobně

- Při načítání/ukládání řetězců do souboru v ’t’ režimu se používá zvolené kódování (defaultně UTF-8 v Pythonu)
- Při zobrazení dochází k další konverzi dle použitého fontu




```
1 t = "k\u016f\u0148"
2 print(len(t))
3 print(t)
```

Možné výsledky podle nastavení IDE/terminálu

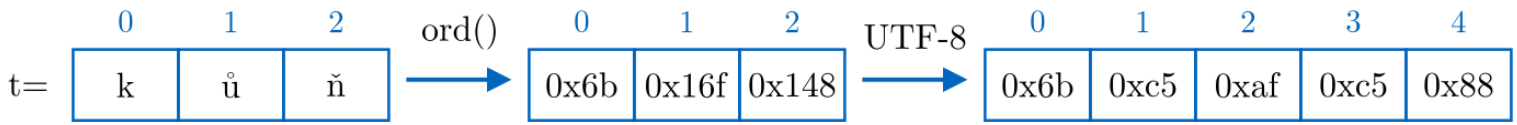
```
3
kůň
```

```
3
k□□
```

Coding podrobně


```
1 t = "k\u016f\u0148"
2 print(len(t))
3 print(t)
```

- Proměnná `t` obsahuje tři UTF-8 znaky:
- `ord`(”k”) = $107_{10} = 6B_{16}$: shodný s pořadí ‘k’ v ASCII, stačí jeden byte
- `ord`(”ů”) = $367_{10} = 16F_{16}$ vnitřní reprezentace dvěma byty (není v ASCII)
- `ord`(”ň”) = $328_{10} = 148_{16}$, vnitřní reprezentace dvěma byty (není v ASCII)
- Proměnná `t` reálně obsahuje 5 bytů



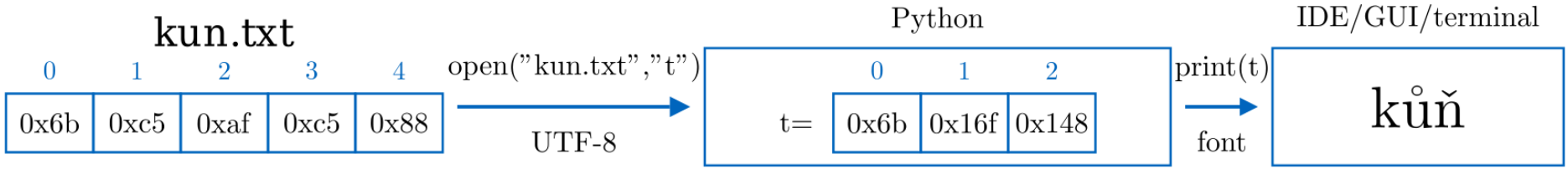

```
1 f = open('kun.txt','wt')
2 f.write(t)
3 f.close()
```

- Soubor ’kun.txt’ obsahuje 5 bytů

Coding podrobně: načtení souboru v režimu ’t’


```
1 f = open('kun.txt','rt')
2 t = f.readline()  #neobsahuje \n, neni treba strip()
3 f.close()
4 print(len(t))
5 print(t)
```

- Při načtení souboru v ‘t’ režimu se (defaultně) použije UTF-8
- Python byty ze souboru interpretuje jako znaky, výsledek je řetězec s třemi znaky



- Výsledné zobrazení ještě ovlivňuje nastavený font

```
3
kůň
```

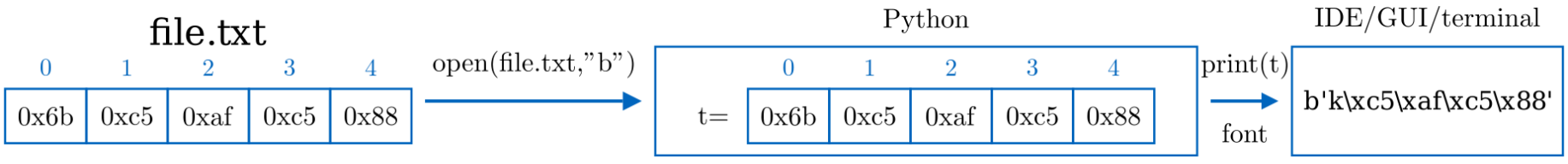
```
3
k□□
```

Coding podrobně: načtení souboru v režimu 'b'


```
1 f = open('kun.txt','rb')
2 t = f.read()  #nacteme vsechny byty
3 f.close()
4 print(len(t))
5 print(t)
6 print(type(t))
```

- Při otevření souboru v 'b' režimu se nepoužívá konverze do UTF-8
- Program načte tolik bytů, kolik je uloženo v souboru
- V proměnné 't' již není text, ale pole bytů: `type(t)= <class 'bytes'>`

```
5
b'k\xc5\xaf\xc5\x88'
<class 'bytes'>
```

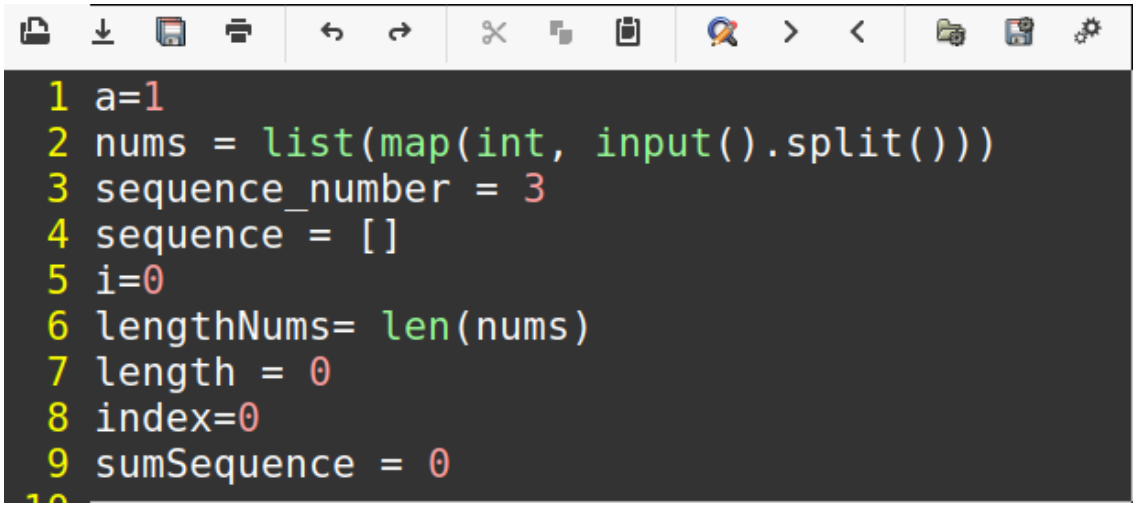


Coding podrobně: BOM v UTF

- BOM — Byte Order Mark je znak “zero width no-break space”
- BOM se vkládá na začátek dat (začátek souboru, streamu, . . .)
- Používá se v UTF-16 pro určení pořadí bytů (endianita)
  - `FE FF` — Big endian
  - `FF FE` — Little endian
- V UTF-8 **nedává smysl!**
- Některé aplikace vkládají na začátek dat tři byty `EF BB BF` pro označení UTF-8
- To usnadňuje detekci typu kódování dat v souboru
- Práce s BOM soubory není všude podporovaná, např. starší Python může spadnout při čtení zdrojového programu s BOM znaky

Coding podrobně: BOM v UTF

- Příklad souboru v UTF-8 s BOM znakem
- Na první pohled je to obyčejný soubor . . .



```
1 a=1
2 nums = list(map(int, input().split()))
3 sequence_number = 3
4 sequence = []
5 i=0
6 lengthNums= len(nums)
7 length = 0
8 index=0
9 sumSequence = 0
```

- Soubor obsahuje BOM sekvenci

```
> file soubor.py
soubor.py: UTF-8 Unicode (with BOM) text, with CRLF line
    terminators
```

Coding podrobně: BOM v UTF

- Načteme v Pythonu, zobrazíme první znak a jeho pořadí v UTF

```
1 f = open("soubor.py", "rt")
2 lineOne = f.readline()
3 print(lineOne[0], ord(lineOne[0]))
```

```
65279
```

- První tři byty jsou převedeny na znak s ord=65279
- print() ovšem nezobrazil vlastní znak!

- Zobrazení všech bytů v souboru. Všimněte si BOM sekvence EF BB BF na začátku

```
00000000 EF BB BF 61 3D 31 0D 0A 6E 75 6D 73 20 3D 20 6C  .  a=1..nums = l
00000010 69 73 74 28 6D 61 70 28 69 6E 74 2C 20 69 6E 70 ist(map(int, inp
00000020 75 74 28 29 2E 73 70 6C 69 74 28 29 29 29 0D 0A ut().split())).
00000030 73 65 71 75 65 6E 63 65 5F 6E 75 6D 62 65 72 20 sequence_number
00000040 3D 20 33 0D 0A 73 65 71 75 65 6E 63 65 20 3D 20 = 3..sequence =
00000050 5B 5D 0D 0A 69 3D 30 0D 0A 6C 65 6E 67 74 68 4E []..i=0..lengthN
00000060 75 6D 73 3D 20 6C 65 6E 28 6E 75 6D 73 29 0D 0A ums= len(nums)..
```

Coding podrobně: BOM v UTF

- Jak se interpretují načtené znaky v Python?

file:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
65279	97	61	49	10	110	117	109	115	32	61	32	108	105	115	116	40	109	97	112	40	105	110	116	44
	a	=	1	\n	n	u	m	s	␣	=	␣	l	i	s	t	(	m	a	p	(	i	n	t	,

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
32	105	110	112	117	116	40	41	46	115	112	108	105	116	40	41	41	41	10	115	101	113	117	101	110
␣	i	n	p	u	t	(	)	·	s	p	l	i	t	(	)	)	)	\n	s	e	q	u	e	n

50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74
99	101	95	110	117	109	98	101	114	32	61	32	51	10	115	101	113	117	101	110	99	101	32	61	32
c	e	–	n	u	m	b	e	r	␣	=	␣	3	\n	s	e	q	u	e	n	c	e	␣	=	␣

75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99
91	93	10	105	61	48	10	108	101	110	103	116	104	78	117	109	115	61	32	108	101	110	40	110	117
[	]	\n	i	=	0	\n	l	e	n	g	t	h	N	u	m	s	=	␣	l	e	n	(	n	u

100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124
109	115	41	10	108	101	110	103	116	104	32	61	32	48	10	105	110	100	101	120	61	48	10	115	117
m	s	)	\n	l	e	n	g	t	h	␣	=	␣	0	\n	i	n	d	e	x	=	0	\n	s	u

125	126	127	128	129	130	131	132	133	134	135	136	137	138	139
109	83	101	113	117	101	110	99	101	32	61	32	48	10	10
m	S	e	q	u	e	n	c	e	␣	=	␣	0	\n	\n

- První tři byty se ze souboru se převedou na znak s ord=65279