

```
270         childpos = rightpos
271         # Move the smaller child up.
272         heap[pos] = heap[childpos]
273         pos = childpos
274         childpos = 2*pos + 1
275     # The leaf at pos is empty now. Put newitem there, and bubble it up
```

Algoritmy a programování

Řetězce, práce se soubory

```
283     # Follow the path to the root, moving parents down until finding a place
284     # newitem fits.
```

```
285     while pos > startpos: Vojtěch Vonásek
```

```
286         parentpos = (pos - 1) >> 1
```

```
287         parent = heap[parentpos]
```

```
288         if parent < newitem: Department of Cybernetics
```

```
289             heap[parentpos] = newitem
290             pos = parentpos
```

```
291             continue
292         break
```

```
293     heap[pos] = newitem
```

```
294
295     def _siftup_max(heap, pos):
296         'Maxheap variant of _siftup'
```

```
297         endpos = len(heap)
```

```
298         startpos = pos
```

```
299         newitem = heap[pos]
```

```
300         # Bubble up the larger child until hitting a leaf.
```

```
301         childpos = 2*pos + 1 # leftmost child position
```

```
302         while childpos < endpos:
```

- 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"))` → 4
 - `print(len("0"))` → 1
 - `print(len(""))` → 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

- 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)
```

- 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
```

- 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
```

- 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
```

- 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
```

- 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
```

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

- Stringy jsou immutable, nelze je měnit přímo přes operátor []
- 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
```

- Stringy jsou immutable, nelze je měnit přímo přes operátor []
- 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
*

```

t[3:]

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

t[:]

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

t[:-3]

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

t[3:-3]

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

t[3:4]

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

t[2:5]

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

t[1:-1]

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

t[4:4]

0	1	2	3	4	5	6	7
a	b	c	d	e	f	g	h

`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 ekvivalentní `print(a,b,c,sep=" ",end="\n")`

- 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
```

```
*
```

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

Doplňte program:

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

aby vypsalo:

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

- 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)
- Řídící znaky (0–31), též netisknutelné znaky
 - Ovlivňují výstup, např. nový řádek (`\n`), tabulátor (`\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 control characters		ASCII printable characters				Extended ASCII characters									
00	NULL (Null character)	32	space	64	@	96	`	128	Ç	160	á	192	Ł	224	Ó
01	SOH (Start of Header)	33	!	65	A	97	a	129	ü	161	í	193	ł	225	ô
02	STX (Start of Text)	34	"	66	B	98	b	130	é	162	ó	194	Ł	226	õ
03	ETX (End of Text)	35	#	67	C	99	c	131	à	163	ù	195	ł	227	ö
04	EOT (End of Trans.)	36	\$	68	D	100	d	132	ä	164	ñ	196	—	228	ø
05	ENQ (Enquiry)	37	%	69	E	101	e	133	à	165	Ñ	197	†	229	ő
06	ACK (Acknowledgement)	38	&	70	F	102	f	134	á	166	ª	198	‡	230	µ
07	BEL (Bell)	39	'	71	G	103	g	135	ç	167	º	199	Ä	231	þ
08	BS (Backspace)	40	(72	H	104	h	136	ê	168	¿	200	Å	232	þ
09	HT (Horizontal Tab)	41)	73	I	105	i	137	ë	169	®	201	Æ	233	Û
10	LF (Line feed)	42	*	74	J	106	j	138	è	170	™	202	Œ	234	Ü
11	VT (Vertical Tab)	43	+	75	K	107	k	139	ï	171	½	203	Ť	235	Ù
12	FF (Form feed)	44	,	76	L	108	l	140	î	172	¼	204	Ŧ	236	Ý
13	CR (Carriage return)	45	-	77	M	109	m	141	í	173	⅓	205	⚡	237	Ŷ
14	SO (Shift Out)	46	.	78	N	110	n	142	Ā	174	¼	206	⚡	238	—
15	SI (Shift In)	47	/	79	O	111	o	143	Ă	175	½	207	⚡	239	·
16	DLE (Data link escape)	48	0	80	P	112	p	144	Ĕ	176	¾	208	⚡	240	≡
17	DC1 (Device control 1)	49	1	81	Q	113	q	145	æ	177	¾	209	⚡	241	±
18	DC2 (Device control 2)	50	2	82	R	114	r	146	Æ	178	¾	210	⚡	242	±
19	DC3 (Device control 3)	51	3	83	S	115	s	147	ø	179	¾	211	⚡	243	¾
20	DC4 (Device control 4)	52	4	84	T	116	t	148	ö	180	¾	212	⚡	244	¶
21	NAK (Negative acknowl.)	53	5	85	U	117	u	149	õ	181	À	213	ı	245	§
22	SYN (Synchronous idle)	54	6	86	V	118	v	150	ü	182	Ā	214	ı	246	÷
23	ETB (End of trans. block)	55	7	87	W	119	w	151	ù	183	Ă	215	ı	247	·
24	CAN (Cancel)	56	8	88	X	120	x	152	ÿ	184	©	216	ı	248	·
25	EM (End of medium)	57	9	89	Y	121	y	153	Ŏ	185	©	217	ı	249	·
26	SUB (Substitute)	58	:	90	Z	122	z	154	Ű	186	©	218	ı	250	·
27	ESC (Escape)	59	;	91	[123	{	155	ø	187	©	219	ı	251	·
28	FS (File separator)	60	<	92	\	124		156	£	188	©	220	ı	252	·
29	GS (Group separator)	61	=	93]	125	}	157	Ø	189	©	221	ı	253	·
30	RS (Record separator)	62	>	94	^	126	~	158	×	190	©	222	ı	254	·
31	US (Unit separator)	63	?	95	_			159	f	191	©	223	ı	255	nbsp
127	DEL (Delete)														

- `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 @
```

- 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: 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  
|
```

- Znaky jsou v počítači uloženy 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
```

	0	1	2	3	4	5	6	7	8
a=	97	98	10	67	68	10	48	49	10
	a	b	\n	C	D	\n	0	1	\n

- 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 nepřidělil body!

MÄ~[to bÄ~[LŽÄ sprÄÄvnÄ~[, ale Brute mi nepL~YidÄ~[lil body!

MΔ~[to bΔ~[EYΓ sprΓ'vnΔ~[, ale Brute mi nepE~YidΔ~[lil body!

- 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 dalšíÈíslo = èíslo + 2
4 print(èíslo, dalšíÈíslo)
```

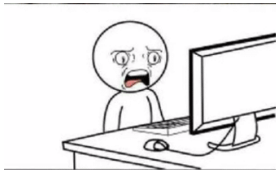
```
File "program8859-2.py", line 1
SyntaxError: Non-UTF-8 code starting with '\xe8' in file program8859-2.py
on line 1, but no encoding declared; see http://python.org/dev/peps/pep-
0263/ for details
```

```

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 podtržítka



```
5 namespace 🟡 = std;
6 using 🟢 = int;
7 using 🟠 = void;
8 using 🕒 = time_t;
9 using 🟡 = bool;
10 #define 🟡 auto
11 #define 🟠 enum
12 #define 🟡 false
13 #define 🟢 true
14 #define 🟠 "evil"
15 #define 🟡 🟡::make_shared
16 #define 🟡 virtual
17 #define 🟡 🟡::cout
18 #define 🟡 🟡::endl
19 template<class 🟡 >
20 using 🟡 = 🟡::vector<🟡>;
21 template<class 🟡 >
22 using 🟡 = 🟡::shared_ptr<🟡>;
23
24 🟡 🟡 { 🟡, 🟡, 🟡, 🟡 };
25 🟡 🟡() { return 🟡::rand(); }
26 🟡 🟡() { return 🟡; }
27
28 struct 🟡 { 🟡 🟡 🟡() = 0; };
29 struct 🟡 : 🟡 { 🟡 🟡 🟡() { 🟡 🟡 "🟡" 🟡 🟡; }; };
30 struct 🟡 : 🟡 { 🟡 🟡 🟡() { 🟡 🟡 "🟡" 🟡 🟡; }; };
31 struct 🟡 : 🟡 { 🟡 🟡 🟡() { 🟡 🟡 "🟡" 🟡 🟡; }; };
32 struct 🟡 : 🟡 { 🟡 🟡 🟡() { 🟡 🟡 "🟡" 🟡 🟡; }; };
33 struct 🟡 : 🟡 { 🟡 🟡 🟡() { 🟡 🟡 "🟡" 🟡 🟡; }; };
34 struct 🟡 : 🟡 { 🟡 🟡 🟡() { 🟡 🟡 "🟡" 🟡 🟡; }; };
35
36 🟡 main()
37 {
38     if (🟡() == 🟡)
39         🟡 << "🟡" << 🟡;
40
41     🟡 << 🟡 >> 🟡 = { 🟡 <🟡 >(), 🟡 <🟡 >(), 🟡 <🟡 >(), 🟡 <🟡 >(), 🟡 <🟡 >() };
42
43     for (🟡 🟡 : 🟡)
44         🟡->🟡();
45
46     return 🟡();
47 }
```

- Soubor je posloupnost bytů

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	25	26	27				
102	105	114	115	116	32	108	105	110	101	10	115	101	99	111	110	100	32	108	105	110	101	10	108	97	115	116	10				
f	i	r	s	t				l	i	n	e	\n	s	e	c	o	n	d				l	i	n	e	\n	l	a	s	t	\n

- 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
```

- 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

	readline() #1										readline() #2							readline() #3										
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	25	26	27
	102	105	114	115	116	32	108	105	110	101	10	115	101	99	111	110	100	32	108	105	110	101	10	108	97	115	116	10
	f	i	r	s	t		l	i	n	e	\n	s	e	c	o	n	d		l	i	n	e	\n	l	a	s	t	\n

```

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
    
```

- 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
```

	line() #1										line() #2										line() #3							
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	25	26	27
	102	105	114	115	116	32	108	105	110	101	10	115	101	99	111	110	100	32	108	105	110	101	10	108	97	115	116	10
	f	i	r	s	t		l	i	n	e	\n	s	e	c	o	n	d		l	i	n	e	\n	l	a	s	t	\n

- Součástí načteného řetězce je konec řádku '\n'

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

names.txt

```
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
```

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

- Proč program nefunguje?

- 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

- 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]
```

	#1				#2				#3				#4		#5		
file:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	49	48	32	10	50	48	32	10	51	48	32	10	52	10	45	52	10
	1	0		\n	2	0		\n	3	0		\n	4	\n	-	4	\n

- 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'`

	0	1	2	3	4	5	6	7	8	9	10	11	12
t:	10	10	115	1111	109	101	10	116	101	120	116	10	10
	\n	\n	s	o	m	e	\n	t	e	x	t	\n	\n

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

	0	1	2	3	4	5	6	7	8	9	10	11	12
t:	10	10	115	1111	109	101	10	116	101	120	116	10	10
	\n	\n	s	o	m	e	\n	t	e	x	t	\n	\n

`t.strip()='some\ntext'`

	0	1	2	3	4	5	6	7	8	9	10	11	12
t:	10	10	115	1111	109	101	10	116	101	120	116	10	10
	\n	\n	s	o	m	e	\n	t	e	x	t	\n	\n


```
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
```

- 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()
```

- 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: ***** 8 , loaded: 0
Saved: ***** 7 , loaded: ***** 8
Saved: ***** 11 , loaded: ***** 7
Saved: ***** 7 , loaded: ***** 11
Saved: ***** 14 , loaded: ***** 7
Saved: ***** 8 , loaded: ***** 14
Saved: ***** 9 , loaded: ***** 8
Saved: ***** 15 , loaded: ***** 9
```

- 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: ***** 15 , loaded: ***** 15
Saved: ***** 9 , loaded: ***** 9
Saved: ***** 17 , loaded: ***** 17
Saved: ***** 17 , loaded: ***** 17
Saved: *** 3 , loaded: *** 3
Saved: ***** 18 , loaded: ***** 18
Saved: ***** 17 , loaded: ***** 17
Saved: ***** 12 , loaded: ***** 12
```

- 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()`

Textový soubor obsahuje postupně:

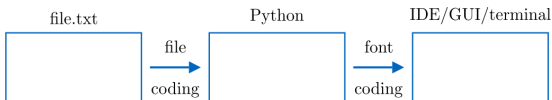
- Jeden řádek s počtem slov n ve skupině 1
- Jeden řádek s počtem slov m ve skupině 2
- n řádků (každý obsahuje jedno slovo)
- m řádků (každý obsahuje jedno slovo)

```
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
```

- 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
```

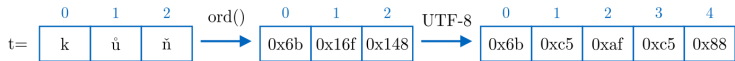
```
3  
k□□
```

```

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

```

- Proměnná `t` obsahuje tři UTF-8 znaky:
- `ord("k") = 10710 = 6B16`: shodný s pořadí 'k' v ASCII, stačí jeden byte
- `ord("ů") = 36710 = 16F16` vnitřní reprezentace dvěma byty (není v ASCII)
- `ord("ň") = 32810 = 14816`, 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()

```

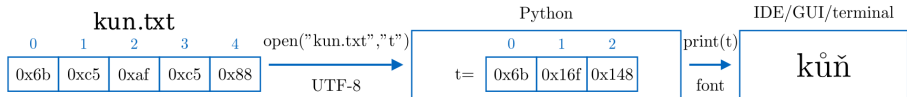
- Soubor `'kun.txt'` obsahuje 5 bytů


```

1 f = open('kun.txt', 'rt')
2 t = f.readline() #neobsahuje \n, není třeba 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□□

```
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 načtení 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'>
```

