Lekce 2

Řetězce. Vstup/výstup.

Řetězce a práce s nimi. Vstup a výstup.

C2184 Úvod do programování v Pythonu
podzim 2016

Stanislav Geidl
Národní centrum pro výzkum biomolekul
Masarykova univerzita

Char (znak)

- libovolný symbol na klavesnici i mimo ni

Regular ASCII Chart (character codes 0 - 127)

000		(nul)	016	▶	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◀	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	–	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009		(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026		(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

Extended ASCII Chart (character codes 128 - 255)

128	Ç	143	Ć	158	×	172	Č	186	║	200	╚	214	Í	228	ń	242	˛
129	ü	144	É	159	č	173	ş	187	╗	201	╔	215	Î	229	ň	243	ˇ
130	é	145	Ĺ	160	á	174	«	188	╝	202	╩	216	ě	230	Š	244	˘
131	â	146	Í	161	í	175	»	189	Ż	203	╦	217	┘	231	š	245	Ş
132	ä	147	ô	162	ó	176	░	190	ż	204	╠	218	┌	232	Ř	246	÷
133	ů	148	ö	163	ú	177	▒	191	┐	205	═	219	█	233	Ú	247	¸
134	ć	149	Ľ	164	Ą	178	▓	192	└	206	╬	220	▄	234	ŕ	248	°
135	ç	150	ľ	165	ą	179	│	193	┴	207	¤	221	Ţ	235	Ű	249	¨
136	ł	151	Ś	166	Ž	180	┤	194	┬	208	đ	222	Ů	236	ý	250	˙
137	ë	152	ś	167	ž	181	Á	195	├	209	Đ	223	▀	237	Ý	251	ű
138	Ő	153	Ö	168	Ę	182	Â	196	─	210	Ď	224	Ó	238	ţ	252	ř
139	ő	154	Ü	169	ę	183	Ě	197	┼	211	Ë	225	ß	239	´	253	ř
140	î	155	Ť	170	¬	184	Ş	198	Ă	212	ď	226	Ô	240	-	254	■
141	Ź	156	ť	171	ź	185	╣	199	ă	213	Ň	227	Ń	241	˝	255	
142	Ä	157	Ł												

- Python nemá speciální typ pro znak, použí pro něj typ `string`
- funkce `chr()` vrací znak pro zadanou ASCII hodnotu
- funkce `ord()` vrací ASCII hodnotu pro zadaný znak (string of length 1)

Speciální znaky, escapovaní

- používá `backslash \`

backslash notace	význam
`\a`	zvonek nebo alert
`\b`	Backspace
`\cx`	Control-x
`\C-x`	Control-x
`\e`	Escape
`\f`	Formfeed
`\M-\C-x`	Meta-Control-x
`\n`	Newline[1]
`\nnn`	kód znaku v osmickove soustave, n = 0..7
`\r`	Carriage return
`\s`	Space
`\t`	Tab
`\v`	Vertical tab
`\x`	Character x
`\xnn`	kód znaku v sestnactkove soustave, n = 0..9,a..f/A..F

[1] zakončení řádku textového souboru záleží na OS

Rozdíly napříč OS a lokací

- základní ASCII kódování 7 bitů (128 znaků), rozšířené kódování 8 bitů (256 znaků)[2]

	Windows	linux	MacOS
nový řádek	\r \n	\n	\n
8-bit. kódování	cp1250 windows-1250	iso-8859-2 latin2	mac_latin2 maccentraleurope

- 16 bitové kodování `utf-8` (Unicode, 65 538 znaků)

Kódování zdrojového souboru (skriptu)

- defaultně `ascii` nebo `utf-8`
- kódování zdrojového souboru (skriptu) mužeme nastavit pomocí na prvním nebo druhém řádku souboru:
  `# -*- coding: windows-1250 -*-`

[2] Přehled standartního kódování: `https://docs.python.org/3/library/codecs.html#standard-encodings`

Řetězec

- je posloupnost znaků
- Python rozpoznává řetězce ohraničené uvozovkami `"` a apostrofy `'`
  `retezec = "ja jsem retezec"`
- řetězce lze spojovat (řetězit) pomocí operatoru `+`
  `"ahoj " + "uzivateli"`
- opakovat operátorem `*`
  `"ahoj " * 10`
- přistupovat ke konkretnímu znaku pomocí indexu nebo podřetězci pomocí rozsahu indexů
  `retezec[0]` nebo `retezec[1:4]`
- na řetězce lze volat vestavěné funkce
  `"ja jsem retezec".find("ja")`
- můžeme zjišťovat délku řetězce
  `len("test")`
- ...

Tvorba a změna řetězce

- nový řetězec vytvoříme například přiřazením[3]

```
retezec1 = 'Ja jsem veta.'
retezec2 = "Ja jsem druha veta."
retezec3 = r"Ja jsem druha veta.\n"
retezec4 = "Ja jsem veliiiiiiiice \
dlouha veta."
retezec5 = """Ja jsem veliiiiiiiice
dlouha veta."""
```

- změnu provedeme libovolným použitím operátoru, např.

```
retezec6 = retezec1 + " " + retezec2
retezec7 = retezec1[:-1] + ", j" +
retezec2[1:]
...
```

[3] v každém příkladu mohou byt uvozovky nahrazeny za apostrof

Přístup k hodnotám

- můžeme přistupovat k jakémukoliv znaku pomocí jeho indexu
  `string[x]`, kde kladná čísla od nuly *n* určují index zleva a zaporná čísla určují index zprava
  ```
  "Danny"[0]
  "Danny"[1]
  "Danny"[-1]
  ```
- přes dvojtečku můžeme nadefinovat rozsah **OD – PO**
  `string[x:y]`, kde tyto výrazy si odpovídají:
  ```
  string[:] == string
  string[x:] == string[x:len(string)]
  string[:y] == string[0:y]
  ```
- pozor na číslovaní! v Pythonu začínáme od nuly!
  ```
  Danny D a n n y
  01234 -5-4-3-2-1
  ```
- co bude výsledkem?

```
"Danny"[1:4]     "Danny"[2:]     "Danny"[:2]
"Danny"[2:2]     "Danny"[-2:]    "Danny"[:-2]
```


Vestavěné funkce pro práci s řetězci I.

- hledání

count

`string1.count(string2)`

`"Danny".count("n")`

vrací počet výskytu `string2` ve `string1`

find

`string1.find(string2)`

`"Danny".find("n")`

vrací index prvního výskytu `string2` ve `string1`

index

`string1.index(string2)`

funguje stejně jako `find`, ale je určen pro kolekce

- nahrazování a rozdělení

replace

`string.replace(old, new[, maxreplace])`

`"Danny".replace("an", "e")`

nahradí `old` za `new` v řetězci `string` maximalně `maxreplace`-krát

split

`string.split(sep)`

`"1 2 3".split(" ")`

vrací list řetězců, které vzniknou rozdělením `string` podle `sep`

Vestavěné funkce pro práci s řetězci II.

- změna velikosti

upper
`string.upper()`
`"danny".upper()`
zvětší všechna písmena

lower
`string.lower()`
`"DANNY".lower()`
zmenší všechna písmena

title
`string.title()`
`"danny je pes".title()`
zvětší první písmena slov

capitalize
`string.capitalize()`
`"danny je pes".capitalize()`
zvětší první písmeno řetězce

swapcase
`string.swapcase()`
`"Danny je Jack Russel".swapcase()`
zamění velikost písmen

Vestavěné funkce pro práci s řetězci III.

- odstraňování "bílých znaků"(␣, `\t`, `\n`, `\r`) na koncích

strip
`string.strip()`
`"\tdanny ".strip()`
odebere bílé znaky z obou konců řetězce

lstrip
`string.lstrip()`
`"\tdanny ".lstrip()`
odebere bílé znaky z levého konce řetězce

rstrip
`string.rstrip()`
`"\tdanny ".rstrip()`
odebere bílé znaky z pravého konce řetězce

- spojování jinak

join
`string.join(collection)`
`", ".join("abcd")`
proloží kolekci řetezcem `string`

Logické operace

```
word = "Hello World"
```

- `word.isalnum()`
  jsou všechny znaky písmena nebo čísla?
- `word.isalpha()`
  jsou všechny znaky písmena?
- `word.isdigit()`
  jsou všechny znaky čísla?
- `word.istitle()`
  jsou všechny písmena slov velká?
- `word.isupper()`
  jsou všechny písmena velká?
- `word.islower()`
  jsou všechny písmena malá?
- `word.isspace()`
  jsou všechny znaky bílé znaky?
- `word.endswith('d')`
  končí řetězec slova/znakem ’d’?
- `word.startswith('H')`
  začíná řetězec slova/znakem ’d’? H
- operátory `in, not in`

Formátování pomocí %

- `"retezec: %s" % promenna`
- formátovací znaky: `%c` znak, `%s` řetězec, `%i` celé číslo, `%f` desetinné číslo (pro booleovskou hodnotu můžeme použít `%s`)
- modifikátory:
  - `-` zarovnání doleva
  - `n`, kde `n` udává celkovou délku
  - `.m`, kde `m` udává počet desetinných míst
- Příklady:

```
"%s" % "Danny"
"%20s" % "Danny"
"%-20s" % "Danny"
```

```
"%f" % 10.3232
"%.2f" % 10.3232
"%10.2f" % 10.3232
```

```
"%s je %s." % ("Danny", "pes")
"%s ma %i roku." % ("Danny", 3)
"%s vazi %.1f kg." % ("Danny", 7.1)
```


Formátování pomocí .format()

- `"retezec: {}".format(promenna)`
- nepovinné formátovací znaky: `{:s}, {:f}, {:d}, ...`[4]
- délka: `{:x}`, kde x je délka
- desetinná část: `{:.y}`, kde y je počet desetinných míst
- zarovnání: `{:<12} {:^12} {:>12}`
- výhoda oproti %, je možnost označit značky čísly 0-9 nebo přímo pojmenovat
- Příklady:

```
"{}".format("Danny")
"{:20}".format("Danny")
"{:>20}".format("Danny")
```

```
"{:f}".format(10.3232)
"{:.2f}".format(10.3232)
"{:10.2f}".format(10.3232)
```

```
"{} je {}.".format("Danny", "pes")
"{1} ma {0} roku.".format(3, "Danny")
"{} vazi {:.1f} kg.".format("Danny", 7.174)
"{0}, {0}, ke mně. Hodnej {0}".format("Danny")
"{j} je {r}.".format(j="Danny",r="J. Russel")
```

[4] Více informací: https://docs.python.org/3.4/library/string.html#format-string-syntax

Vstupy a výstupy

Vstup:

- vstup uživatele `input()` (v Pythonu 2.7 `raw_input()`[5])

```
input()
data = input("Insert number: ")
```

- argumenty programu (přes modul `sys`)
- ze souboru

Výstup:

- print()
- `sys.stdout` a `sys.stderr`
- do souboru

[5] funkce input() v Pythonu 2.7 automaticky převádí zadaný vstup na číslo apod.

Příklad I

```
r = "Danny je Jack Russel."
```

1. převeďte ’je’ na ’Je’ (’Danny Je Jack Russel.’)
2. zachovejte pouze první velké pismeno řetězce r (’Danny je jack russel.’)
3. nahraďte ’Jack Russel’ za ’foxterier’ (’Danny je foxterier.’)
4. najděte indexy x a y, tak aby r[x:y] odpovídalo řetezci "Jack"(’[9:13]’)
5. CHALLENGE: nahraďte první výskyt zadaného znaku za znak napsaný velkými písmeny (např. pro ’s’: ’Danny je Jack RuSsel.’)

Příklad I - řešení

```
r = "Danny je Jack Russel."
```

1. převeďte ’je’ na ’Je’ (’Danny Je Jack Russel.’)

```
r1 = r.title()
r1 = r.replace("je","Je")
```

2. zachovejte pouze první velké pismeno řetězce r (’Danny je jack russel.’)

```
r2 = r.capitalize()
```

3. nahraďte ’Jack Russel’ za ’foxterier’ (’Danny je foxterier.’)

```
r3 = r.replace("Jack Russel","foxterier")
```


Příklad I - řešení (pokračování)

```
r = "Danny je Jack Russel."
```

4. najděte indexy x a y, tak aby r[x:y] odpovídalo řetezci "Jack"('[9:13]')

```
word = "Jack"
start = r.index(word)
end = start + len(word)
print("[{}:{}]".format(start,end))
print(r[9:13])
```

5. CHALLENGE: nahraďte první výskyt zadaného znaku za znak napsaný velkými písmeny (např. pro 's': 'Danny je Jack RuSsel.')

```
char = "s"
i = r.index(char)
r_challenge = r[:i] + r[i].upper() + r[i+1:]
r_challenge = r.replace(char,char.upper(),1)
```


Příklad II

Pomocí funkce `input()` (případně funkce `raw_input()` v Pythonu 2.7) získejte od uživatele tři proměnné `a`, `b` a `c`:

1. zjistěte, jestli se jedná ve všech případech o čísla (např. 'Promenna a je cislo: False')
2. zjistěte, jestli je číslo `b` sudé (např. 'Promenna b je suda: 0', kde 0 je sudé, 1 liché ;))
3. spočítejte celočíselný podíl proměnných `b` a `c` ($\frac{b}{c}$)
4. spočítejte kořeny `x1` a `x2` kvadratické rovnice $ax^2 + bx + c = 0$

Příklad II - řešení

Pomocí funkce `input()` (případně funkce `raw_input()` v Pythonu 2.7) získejte od uživatele tři proměnné `a`, `b` a `c`:

```
a = input()
b = input()
c = input()
```

1. zjistěte, jestli se jedná ve všech případech o čísla (např. 'Promenna a je cislo: False')

```
print("Pr. a je cislo: {}".format(a.isdigit()))
print("Pr. b je cislo: {}".format(b.isdigit()))
print("Pr. c je cislo: {}".format(c.isdigit()))
```

2. zjistěte, jestli je číslo `b` liché (např. 'Promenna b je licha: True')

```
b = int(b)
print("Pr. b je liche cislo: {}".format(bool(b%2)))
```


Příklad II - řešení (pokračování)

Pomocí funkce `input()` (případně funkce `raw_input()` v Pythonu 2.7) získejte od uživatele tři proměnné `a`, `b` a `c`:

```
a = input()
b = input()
c = input()
```

4. spočítejte kořeny `x1` a `x2` kvadratické rovnice $ax^2 + bx + c = 0$

```
a = int(a)
D = b**2 - 4*a*c
x1 = (-b+D**.5) / (2*a)
x2 = (-b-D**.5) / (2*a)
print(x1)
print(x2)
```