Lekce 3

Základy programování II.

Pokročilé datové typy - řetězce a kolekce (tuple, seznamy, slovníky).

C2184 Úvod do programování v Pythonu
podzim 2014

Mgr. Stanislav Geidl
Národní centrum pro výzkum biomolekul
Masarykova univerzita

Char (znak)

- libovolný znak 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	Ž	203	╦	217	┘	231	š	245	Ş
132	ä	147	ô	162	ó	176	░	190	ž	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 string, ale má pro ně vestavěné funkce
- funkce `chr()` vrací znak pro zadanou ASCII hodnotu
- funkce `ord()` vrací ASCII hodnotu pro zadaný znak

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..7,a..f/A..F

[1] zakončení řádku textového souboru záleží na OS

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[2]

```
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:]
...
```

[2] 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
  `string[x:y]`, kde tyto výrazy si odpovídají:
  ```
  string[:] == string
  string[x:] == string[x:len(string)+1]
  string[:y] == string[0:y]
  ```
- pozor na číslovaní! v Pythonu začínáme od nuly!
  ```
  Danny
  01234
  ```
- 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)`

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

nahradí `old` za `new` v řetězci `string`

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()
```
zvětší první písmena slov

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 čísla?
- `word.isalpha()`
  jsou všechny znaky písmena?
- `word.isdigit()`
  obsahuje řetězec čísla?
- `word.istitle()`
  obsahuje řetězec titulky (slova s prvním velkým písmenem)?
- `word.isupper()`
  obsahuje řetězec slova s velkými písmeny?
- `word.islower()`
  obsahuje řetězec slova s malými písmeny?
- `word.isspace()`
  obsahuje řetězec bílé znaky?
- `word.endswith('d')`
  končí řetězec slova/znakem 'd'?
- `word.startswith('H')`
  začíná řetězec slova/znakem 'd'? H
- opakování: operátory `in, not in`

Formátování pomocí %

- `"retezec: %s" % promenna`
- formátovací znaky: `%c` znak, `%s` řetězec, `%i` celé číslo, `%f` desetinné číslo
- modifikátory:
  - `-` zarovnání doleva
  - `n`, kde n udává celkovou délku
  - `.n`, kde n 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()

- test

List (seznam)

- patří do kolecí, podobně jako N-tice a slovník
- vytváříme pomocí hranatých závorek `[]`
  `["a", "b", "c", "d"]`
- každý prvek má svůj automatický index, který odpovídá pořadí

Práce se seznamy I.

- vytvoření

```
seznam1 = [1, 1, 2, 3, 5, 8, 13]
seznam2 = list(seznam1)
seznam3 = seznam1[:]
seznam4 = seznam1 # nejedna se o novy list,
pouze odkaz na stary!!!
seznam5 = range(2,20,2)
# [2, 4, 6, 8, 10, 12, 14, 16, 18]
```

- přidáváme prvky

```
seznam1.append(21)
# [1, 1, 2, 3, 5, 8, 13, 21]
seznam2.insert(2, 90)
# [1, 1, 90, 2, 3, 5, 8, 13]
seznam3.extend([21, 34])
# [1, 1, 90, 2, 3, 5, 8, 13, 21, 34]
seznam3.append([21, 34])
# [1, 1, 2, 3, 5, 8, 13, [21, 34]]
```


Práce se seznamy II.

- mazání

```
seznam1.remove(1)
# [1, 2, 3, 5, 8, 13, 21]
last = seznam1.pop()[3]
# last = 21; [1, 1, 2, 3, 5, 8, 13]
```

- výběr pomocí [], stejně jako u řetězců

```
seznam1[1]
seznam1[1:5]
seznam1[-2:]
```

- přehození směru

```
seznam1.reverse()
# [13, 8, 5, 3, 2, 1, 1]
```

- vyhledávání

```
seznam1.index(5) # 4
seznam1.count(1) # 2
```

- seřazení

```
seznam = [1, 4, 3, 6, 2, 5]
seznam.sort() # [1, 2, 3, 4, 5, 6]
seznam.sort(reverse=True) # [6, 5, ...
```

[3] list funguje jako zásobník, FIFO = first in, first out

Práce se seznamy III.

- počítání

```
seznam1 = [1, 1, 2, 3, 5, 8, 13]
len(seznam1) # 7
sum(seznam1) # 33
min(seznam1) # 1
max(seznam1) # 13
```

- procházení

```
for item in [1, 2, 3]:
  print(item)
for item in range(1,4):
  print(item)
```


Tuple (N-tice)

- vytváříme pomocí jednoduchých závorek `()`
- můžeme s nimi pracovat podobně jako se seznamy, jenom je nemužeme měnit, tzn. že funkce `append` a další nejsou dostupné
- můžeme jednoduše převádět na list pomocí `list((1,2))` a podobně zpět `tuple([1,2])`

Dictionary (slovník)

- vytváříme pomocí složených závorek `{}`
  `{1: 3, 2: 4}`
- prvek ve slovníku se skládá z klíče a jeho hodnoty, 1 a 2 jsou klíče, jejich hodnoty jsou 3, resp. 4
- nefungují zde indexy, na hodnoty se dotazujeme pomocí klíče
- každý klíč je unikatní, žádný slovník nemůže obsahovat dva stejné klíče

Práce se seznamy

- vytvoření
```
dict = {'Name': 'Zara', 'Age': 7, 'Class':
'First'}
```
- čtení
```
print(dict['Name']) # Zara
```
- úprava hodnot
```
dict['Age'] = 8 # úprava stávající hodnoty
dict['School'] = "DPS School"# přidání nové
```
- smazání hodnot
```
del dict['Name']
dict.clear() # smaže všechny položky
del dict # smaže celý slovník
```
- procházení hodnot
```
for key in dict:
  print(key)
  print(dict[key])
```

Vícerozměrné kolekce

- kolekce můžeme kombinovat a vytvářet list listů, ...
  `[[1, 2], [2, 3], [4, 5]]`
- můžeme kombinovat i navzájem a vytvářet list N-tic, ...
  `[(1, 2), (2, 3), (4, 5)]`