Strings

Strings: type `str`

non-scalar, compound types:
composed of accessible small pieces

strings: ordered **sequence** of characters (sequence type)
homogeneous: all items are single characters (same type)

literals: quoted (single or double)
`'abc-475', 'Tomorrow', ''Hanna's bag'', 'aa@aaa.aa'`

operators (block): + (concatenation), * (replication)
function `len` (standard library)

comparison operators: lexicographical order (ASCII table)

```
>>> 'abc' + '-475'
'abc-475'
>>> len ('measures')
8
>>> 'abc'*3
'abcabcabc'
>>> 'a' < 'v'
True
>>>'7' > '9'
False
>>> 'mouth' < 'nose'
True
>>> 'ambassador'>'amber'
False
>>>'100' < '200'
True
>>> '100' < '2'
True
```

ASCII table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

ASCII: American Standard Code for Information Interchange

Useful intervals:

- [‘0’, ‘9’]
- [‘A’, ‘Z’]
- [‘a’, ‘z’]

Special characters

void character: ‘’ (there is nothing enclosed)

SPACE character: ‘ ‘ (a space enclosed)

line feed character: ‘`\n`’ (new line,
1 character, not 2)

All of them have its corresponding ASCII code

«notice» the space and the line feed characters

```
>>> nothing = ''
>>> space = ' '
>>> linefeed = '\n'
>>> len(nothing)
0
>>> len(space)
1
>>> len(linefeed)
1
>>> a = 'apples'
>>> b = 'cherries'
>>> c = 'oranges'
>>> x = a + ' '+ b + '\n' + c
>>> print(x)
apples cherries
oranges
>>> len(x)
23
>>> len(a) + len(b) + len(c) + 2
23
```

Indexing

indexing operator: `[]` (square brackets)

index: integer expression. Character **position**

Let `s` be a string:
`s[index]` character of `s` at position `index`

positive, **zero-based** indices: 0 <= index < len(s)
negative indices: -len(s) <= index < 0

```
>>> s = 'strawberry'
>>> s[4]
'w'
```

characters:	s	t	r	a	w	b	e	r	r	y
positive indices:	0	1	2	3	4	5	6	7	8	9
negative indices:	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Indexing

first character

last character

(recommended syntax)

positive indexing

negative indexing

len(a) is 10: indexes range is 0..9
index 10 is out of range
Error: `string index out of range`

```
>>> a = 'clean room'
>>> len(a)
10
>>> a[0]
'c'
>>> a[len(a)-1]
'm'
>>> a[-1]
'm'
>>> a[5]
' '
>>> a[1]
'l'
>>> a[-2]
'o'
>>> a[-10]
'c'
>>> a[10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Slicing

slice: substring
slicing operator: `[start:stop:step]`

start, stop, step: integer expressions

Let `s` be a string:
`s[start:stop]` substring of `s` from `start` (included) to `stop` (not included)

positive indices: 0 <= start < stop < len(s)

`step:` jumps step characters

Default values:
start: 0
stop: len(s)
step: 1

```
>>> s = 'strawberry'
>>> s[2:5]
'raw'
>>> a[1:len(s):2]
'tabry'
```

Slicing

Default values:
start: 0
stop: len(s)
step: 1

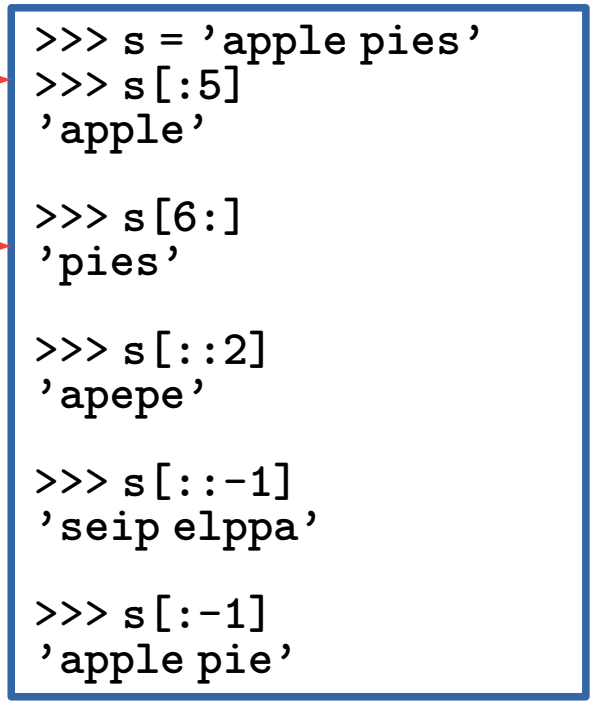
default start: 0
default step: 1

default stop: len(s)
default step: 1

default start: 0
default stop: len(s)

reversing

all but the last



Membership operators

operators: `in`
`not in`
result: `Boolean`

a string is a substring of itself

void string is a substring of any other string

definition of a set of characters

check for membership

```
>>> s = 'apple pies'

>>> 'pie' in s
True
>>> 'banana' in s
False
>>> s in s
True
>>> '' in s
True

>>> my_name = 'arya stark'
>>> 'a' in my_name
True
>>> 'e' in my_name
False
```

String immutability

Strings are immutable: cannot be modified

string `s` cannot be modified

name `s` now refers to a different string expression: string `s` has been redefined, refers to another value

```
>>>> s = 'lemons'

>>> s[0] = 'd'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment

>>> s = 'd' + s[1:]
>>> s
'demons'
```

Strings: built-in functions

functions: `len`, `max`, `min`

string length

maximum character (*)

minimum character (*)

maximum string (*)

minimum string (*)

(*) lexicographical order

```
>>> len('telecommunications')
18
>>> max('telecommunications')
'u'
>>> min('telecommunications')
'a'
>>> max('method', 'you', 'aerial', 'brown', 'apple')
'you'
>>> min('method', 'you', 'aerial', 'brown', 'apple')
'aerial'
```

Strings: `str` methods

method: function that applies to a determined type object

`str` methods: methods applicable to `str` objects
invoking (calling) a method syntax: **dot notation**

```
str_variable_name.str_method_name(actual_parameters)
```

Lists all methods

Short documentation through the shell
Press q to quit

```
>>> dir(str)
[...'capitalize', 'casefold', 'center', 'count',
'encode', 'endswith', 'expandtabs', 'find', ...]
>>> help(str.count)
Help on method_descriptor:
count(...)
  S.count(sub[, start[, end]]) -> int

  Return the number of non-overlapping occurrences of
  substring sub in string S[start:end]. Optional arguments
  start and end are interpreted as in slice notation.
```

`str` method: `count`

```
>>> help(str.count)

  S.count(sub[, start[, end]]) -> int

  Return the number of non-overlapping occurrences of
  substring sub in string S[start:end]. Optional arguments
  start and end are interpreted as in slice notation.

>>>a = 'tomorrow'
>>> a.count('o')
3
>>> 'banana'.count('na')
2
>>> 'aaaaaa'.count('aa')
3
```

non-overlapping: →

`str` method: `find`

indexing inverse process

```
>>> help(str.find)

  S.find(sub[, start[, end]]) -> int

  Return the lowest index in S where substring sub is found,
  such that sub is contained within S[start:end]. Optional
  arguments start and end are interpreted as in slice notation.

  Return -1 on failure.

>>> 'banana'.find('anana')
1
>>> 'banana'.find('n')
2
>>> 'banana'.find('cherry')
-1
```

index of first '`n`'

not found: -1

`str` method: `replace`

returns a new string

```
>>> help(str.replace)
  S.replace(old, new[, count]) -> str

  Return a copy of S with all occurrences of substring
  old replaced by new.  If the optional argument count is
  given, only the first count occurrences are replaced.

>>> 'lemon'.replace('l', 'd')
'demon'
```

successive replacements →

```
>>> s1 = 'banana'
>>> s2 = s1.replace('a', 'o')
>>> s3 = s2.replace('non', 'nob')
>>> s1, s2, s3
('banana', 'bonono', 'bonobo')
```

2 replacements →

```
>>> 'bad bad bad bad'.replace('bad', 'good', 2)
'good good bad bad'
```

`str` methods: `isalpha, ...`

All of them return a `bool` `True` if

`isalnum:` all characters are alphanumeric
`isalpha:` all characters are alphabetic
`isdigit`: all characters are digits
`islower`: all cased characters are lowercase
and there is at least one cased character
…

and return `False` otherwise

```
>>> 'abc989'.isalpha()
False
>>> 'abc989'.isalnum()
True
>>> 'abc989'.isdigit()
False
>>> 'abc989'.islower()
True
>>> '4455'.islower()
False
>>> '4455'.isupper()
False
>>> '4455'.isdigit()
True
>>>'abcDEF989'.islower()
False
>>> 'abcDEF989'.isupper()
False
>>> 'DEF989'.isupper()
True
```

`str` methods to switch case

return a new string.

`lower, upper, swapcase`

`capitalize, title`

```
>>> a = 'How are you'
>>> b = 'Programming'
>>> c = 'USA'
>>> a.swapcase()
'hOW ARE YOU'
>>> c.lower()
'usa'
>>> b.upper()
'PROGRAMMING'
>>> d = 'albert'
>>> d.capitalize()
'Albert'
>>> e = 'dr. albert fox'
>>> e.title()
'Dr. Albert Fox'
```

`str` methods: miscellany

return a new string:

`ljust, center, rjust, zfill`

```
>>> a = 'Hello !!!'
>>> a.ljust(20, ' ')
'Hello !!!      '
>>> a.center(20, ' ')
'   Hello !!!   '
>>> a.rjust(20, ' ')
'      Hello !!!'
>>> a.ljust(20, '-')
'Hello !!!----------'
>>> '987'.zfill(6)
'000987'
```

return a Boolean:

`endswith, startswith`

equivalent expressions

```
>>> b = 'telecommunication'
>>> prefix = 'tele'
>>> suffix = 'tion'
>>> b.startswith(prefix)
True
>>> b[:len(prefix)] == prefix
True
>>> b.endswith(suffix)
True
>>> b[-len(suffix):] == suffix
True
```