

61A Lecture 10

Wednesday, September 21

Strings are an Abstraction

Strings are an Abstraction

Representing data:

'200'

'1.2e-5'

'False'

'(1, 2)'

Strings are an Abstraction

Representing data:

'200' '1.2e-5' 'False' '(1, 2)'

Representing language:

```
""""0! methinks how slow  
This old moon wanes; she lingers my desires ,  
Like to a step dame, or a dowager  
Long withering out a young man's revenue.""""
```

Strings are an Abstraction

Representing data:

'200' '1.2e-5' 'False' '(1, 2)'

Representing language:

```
""""0! methinks how slow
This old moon wanes; she lingers my desires ,
Like to a step dame, or a dowager
Long withering out a young man's revenue.""""
```

Representing programs:

```
'curry = lambda f: lambda x: lambda y: f(x, y)'
```

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
3 bits	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
	1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
	2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
	6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

3 bits

4 bits

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

3 bits

4 bits

- Layout was chosen to support sorting by character code

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

3 bits

4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

3 bits

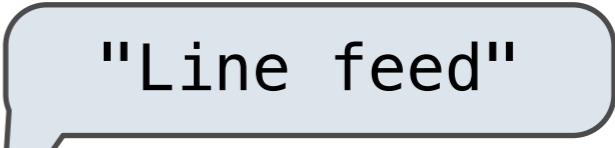
4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

ASCII Code Chart



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

3 bits

4 bits

- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

ASCII Code Chart

The chart is a 16x8 grid. The columns are labeled 0 through F. The rows are labeled 0 through 7. A blue bracket on the left side spans rows 2 through 7 and is labeled '3 bits'. A blue bracket at the bottom spans all 8 columns and is labeled '4 bits'. Two callout boxes are present: one pointing to the 'BEL' cell (row 0, column 7) labeled '"Bell"' and another pointing to the 'LF' cell (row 0, column 10) labeled '"Line feed"'. The cells in the first two rows (0 and 1) are shaded pink. The cell 'DEL' in the last row (7, column F) is also shaded pink.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL

- Layout was chosen to support sorting by character code
- Rows indexed 2–5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission

Representing Strings: the Unicode Standard

**Bonus
Material**

Representing Strings: the Unicode Standard

聾	聾	聾	聽	聵	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腓	腳	腓	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
萵	萵	荳	菰	葱	苳	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

Representing Strings: the Unicode Standard

- 109,000 characters

聾	聾	聾	聽	聵	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腭	腳	腴	暇	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菘	菘	荳	菝	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)

聾	聾	聾	聽	聵	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腓	腳	腓	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菘	菘	荳	菘	葱	苘	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case

聾	聾	聾	聽	聵	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腭	腳	腴	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菘	菘	荳	菝	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order

聾	聾	聾	聽	聵	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腭	腳	腴	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菘	菘	荳	菰	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- 32 bits per character number

聾	聾	聾	聽	聵	聵	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腓	腳	腓	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菘	菘	荳	菘	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- 32 bits per character number
- A canonical name for every character

聾	聾	聾	聽	聵	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腓	腳	腓	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菘	菘	荳	菘	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- 32 bits per character number
- A canonical name for every character

聾	聾	聾	聽	聵	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腓	腳	腓	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菘	菘	荳	菘	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

U+0058 LATIN CAPITAL LETTER X

Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- 32 bits per character number
- A canonical name for every character

聾	聾	聾	聽	聵	聵	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腓	腳	腓	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菘	菘	荳	菘	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- 32 bits per character number
- A canonical name for every character

聾	聾	聾	聽	聵	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腓	腳	腓	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菘	菘	荳	菘	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE

Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- 32 bits per character number
- A canonical name for every character

聾	聾	聾	聽	聵	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腩	腳	腴	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菀	菀	荳	菴	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菀	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE



Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- 32 bits per character number
- A canonical name for every character

聾	聾	聾	聽	聵	聵	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腓	腳	腓	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菟	菟	荳	菝	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE



Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- 32 bits per character number
- A canonical name for every character

聾	聾	聾	聽	聵	聵	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	腓	腳	腓	股	股	膈	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艷	艷	艷	艷	艷	艸
8271	8272	8273	8274	8275	8276	8277	8278
菘	菘	荳	菘	葱	苜	荷	葶
8371	8372	8373	8374	8375	8376	8377	8378
葱	菘	葳	葳	葵	葶	葶	蔥

http://ian-albert.com/unicode_chart/unichart-chinese.jpg

U+0058 LATIN CAPITAL LETTER X

U+263a WHITE SMILING FACE

U+2639 WHITE FROWNING FACE



Demo

Representing Strings: UTF-8 Encoding

**Bonus
Material**

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits, and can encode any integer 0-255

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits, and can encode any integer 0–255

bytes

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits, and can encode any integer 0–255

bytes

integers

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits, and can encode any integer 0–255

00000000

0

bytes

integers

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits, and can encode any integer 0-255

	00000000	0	
bytes	00000001	1	integers

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits, and can encode any integer 0–255

	00000000	0	
bytes	00000001	1	integers
	00000010	2	

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits, and can encode any integer 0–255

	00000000	0	
bytes	00000001	1	integers
	00000010	2	
	00000011	3	

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits, and can encode any integer 0–255

	00000000	0	
bytes	00000001	1	integers
	00000010	2	
	00000011	3	

Variable-length encoding: integers vary in the number of bytes required to encode them!

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits, and can encode any integer 0-255

	00000000	0	
bytes	00000001	1	integers
	00000010	2	
	00000011	3	

Variable-length encoding: integers vary in the number of bytes required to encode them!

In Python: `string` length in characters, `bytes` length in bytes

Representing Strings: UTF-8 Encoding

UTF: (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and 32 bit numbers

UTF-8: Correspondence between numbers and bytes

A byte is 8 bits, and can encode any integer 0–255

	00000000	0	
bytes	00000001	1	integers
	00000010	2	
	00000011	3	

Variable-length encoding: integers vary in the number of bytes required to encode them!

In Python: `string` length in characters, `bytes` length in bytes

Demo

Strings are Sequences

Strings are Sequences

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

Strings are Sequences

```
>>> city = 'Berkeley'  
>>> len(city)  
8  
>>> city[3]  
'k'
```

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

Strings are Sequences

```
>>> city = 'Berkeley'
```

```
>>> len(city)
```

```
8
```

```
>>> city[3]  
'k'
```

An element of a string
is itself a string!

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

Strings are Sequences

```
>>> city = 'Berkeley'
```

```
>>> len(city)
```

```
8
```

```
>>> city[3]  
'k'
```

An element of a string
is itself a string!

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

```
>>> 'Berkeley' + ', CA'
```

```
'Berkeley, CA'
```

```
>>> 'Shabu ' * 2
```

```
'Shabu Shabu '
```

Strings are Sequences

```
>>> city = 'Berkeley'
```

```
>>> len(city)
```

```
8
```

```
>>> city[3]  
'k'
```

An element of a string
is itself a string!

Length. A sequence has a finite length.

Element selection. A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0 for the first element.

```
>>> 'Berkeley' + ', CA'
```

```
'Berkeley, CA'
```

```
>>> 'Shabu ' * 2
```

```
'Shabu Shabu '
```

String arithmetic is
like tuple arithmetic

String Membership Differs from Other Sequences

String Membership Differs from Other Sequences

The "in" and "not in" operators match substrings

String Membership Differs from Other Sequences

The "in" and "not in" operators match substrings

```
>>> 'here' in "Where's Waldo?"  
True
```

String Membership Differs from Other Sequences

The "in" and "not in" operators match substrings

```
>>> 'here' in "Where's Waldo?"  
True
```

Why? Working with strings, we care about words, not characters

String Membership Differs from Other Sequences

The "in" and "not in" operators match substrings

```
>>> 'here' in "Where's Waldo?"  
True
```

Why? Working with strings, we care about words, not characters

The "count" method also matches substrings

String Membership Differs from Other Sequences

The "in" and "not in" operators match substrings

```
>>> 'here' in "Where's Waldo?"  
True
```

Why? Working with strings, we care about words, not characters

The "count" method also matches substrings

```
>>> 'Mississippi'.count('i')  
4  
>>> 'Mississippi'.count('issi')  
1
```

String Membership Differs from Other Sequences

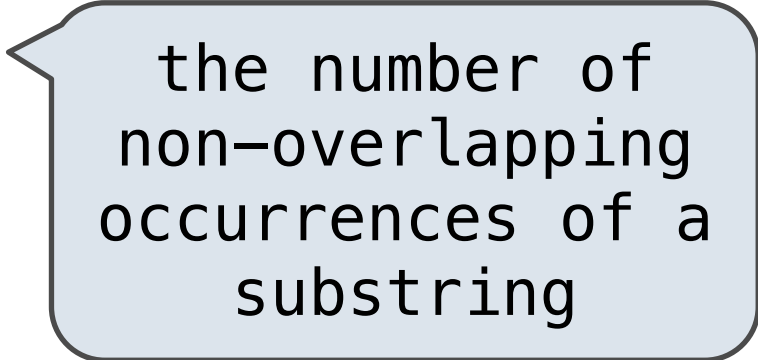
The "in" and "not in" operators match substrings

```
>>> 'here' in "Where's Waldo?"  
True
```

Why? Working with strings, we care about words, not characters

The "count" method also matches substrings

```
>>> 'Mississippi'.count('i')  
4  
>>> 'Mississippi'.count('issi')  
1
```



the number of
non-overlapping
occurrences of a
substring

String Membership Differs from Other Sequences

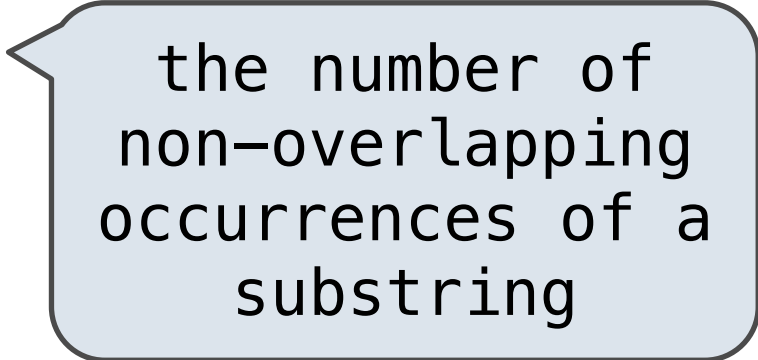
The "in" and "not in" operators match substrings

```
>>> 'here' in "Where's Waldo?"  
True
```

Why? Working with strings, we care about words, not characters

The "count" method also matches substrings

```
>>> 'Mississippi'.count('i')  
4  
>>> 'Mississippi'.count('issi')  
1
```



the number of
non-overlapping
occurrences of a
substring

String Literals Have Three Forms

```
>>> 'I am string!'  
'I am string!'
```

```
>>> "I've got an apostrophe"  
"I've got an apostrophe"
```

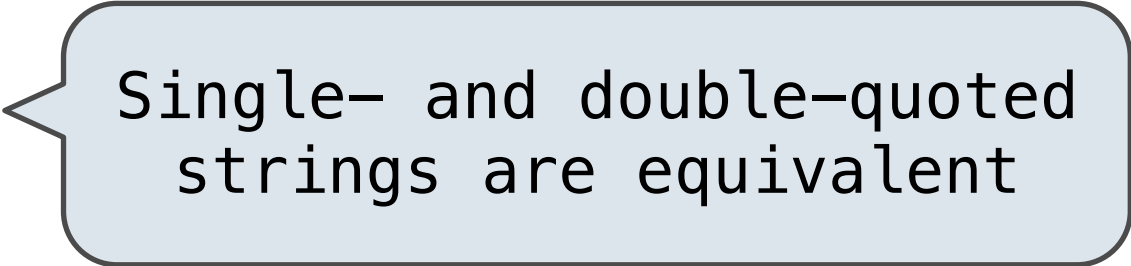
```
>>> '您好'  
'您好'
```


String Literals Have Three Forms

```
>>> 'I am string!'
'I am string!'
```

```
>>> "I've got an apostrophe"
"I've got an apostrophe"
```

```
>>> '您好'
'您好'
```



Single- and double-quoted strings are equivalent

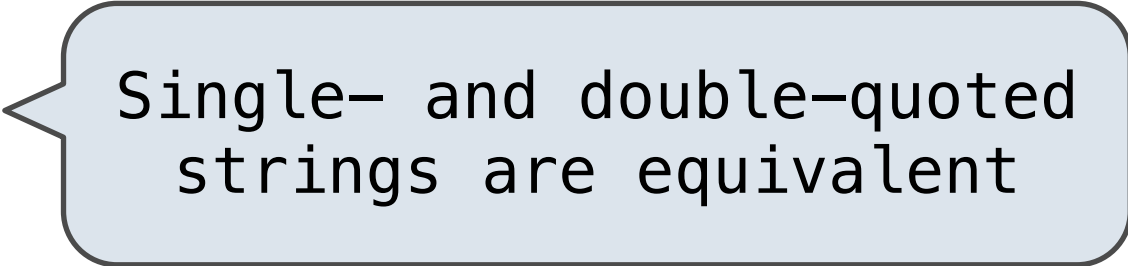
String Literals Have Three Forms

```
>>> 'I am string!'
'I am string!'
```

```
>>> "I've got an apostrophe"
"I've got an apostrophe"
```

```
>>> '您好'
'您好'
```

```
>>> """The Zen of Python
claims, Readability counts.
Read more: import this."""
'The Zen of Python\nclaims, "Readability counts."\nRead more:
import this.'
```



Single- and double-quoted strings are equivalent

String Literals Have Three Forms

```
>>> 'I am string!'
'I am string!'
```

```
>>> "I've got an apostrophe"
"I've got an apostrophe"
```

```
>>> '您好'
'您好'
```

```
>>> """The Zen of Python
claims, Readability counts.
Read more: import this."""
'The Zen of Python\nclaims, "Readability counts."\nRead more:
import this.'
```

Single- and double-quoted strings are equivalent

A backslash "escapes" the following character

String Literals Have Three Forms

```
>>> 'I am string!'
'I am string!'
```

```
>>> "I've got an apostrophe"
"I've got an apostrophe"
```

```
>>> '您好'
'您好'
```

```
>>> """The Zen of Python
claims, Readability counts.
Read more: import this."""
'The Zen of Python\nclaims, "Readability counts."\nRead more:
import this.'
```

Single- and double-quoted strings are equivalent

A backslash "escapes" the following character

"Line feed" character represents a new line

String Coercion

String Coercion

Any object can be "coerced" into a string.

String Coercion

Any object can be "coerced" into a string.

Coercion doesn't change an object; it produces a corresponding object of a different type.

String Coercion

Any object can be "coerced" into a string.

Coercion doesn't change an object; it produces a corresponding object of a different type.

```
>>> digits
```


String Coercion

Any object can be "coerced" into a string.

Coercion doesn't change an object; it produces a corresponding object of a different type.

```
>>> digits  
(1, 8, 2, 8)
```

String Coercion

Any object can be "coerced" into a string.

Coercion doesn't change an object; it produces a corresponding object of a different type.

```
>>> digits  
(1, 8, 2, 8)
```

```
>>> 2 in digits
```

String Coercion

Any object can be "coerced" into a string.

Coercion doesn't change an object; it produces a corresponding object of a different type.

```
>>> digits  
(1, 8, 2, 8)
```

```
>>> 2 in digits  
True
```

String Coercion

Any object can be "coerced" into a string.

Coercion doesn't change an object; it produces a corresponding object of a different type.

```
>>> digits  
(1, 8, 2, 8)
```

```
>>> 2 in digits  
True
```

```
>>> str(2) + ' is an element of ' + str(digits)
```

String Coercion

Any object can be "coerced" into a string.

Coercion doesn't change an object; it produces a corresponding object of a different type.

```
>>> digits  
(1, 8, 2, 8)
```

```
>>> 2 in digits  
True
```

```
>>> str(2) + ' is an element of ' + str(digits)  
'2 is an element of (1, 8, 2, 8)'
```

String Coercion

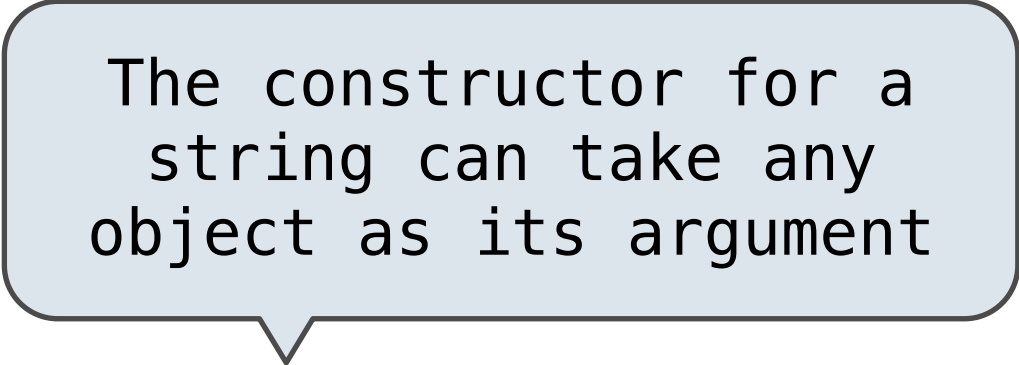
Any object can be "coerced" into a string.

Coercion doesn't change an object; it produces a corresponding object of a different type.

```
>>> digits  
(1, 8, 2, 8)
```

```
>>> 2 in digits  
True
```

```
>>> str(2) + ' is an element of ' + str(digits)  
'2 is an element of (1, 8, 2, 8)'
```



The constructor for a string can take any object as its argument

String Coercion

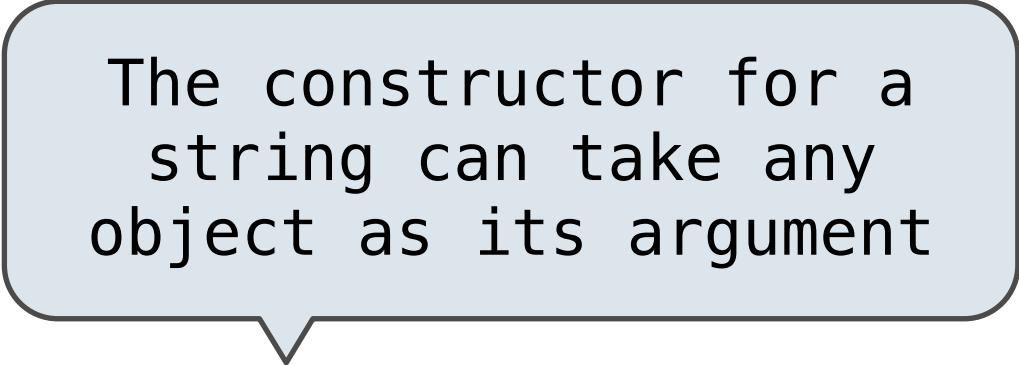
Any object can be "coerced" into a string.

Coercion doesn't change an object; it produces a corresponding object of a different type.

```
>>> digits  
(1, 8, 2, 8)
```

```
>>> 2 in digits  
True
```

```
>>> str(2) + ' is an element of ' + str(digits)  
'2 is an element of (1, 8, 2, 8)'
```



The constructor for a string can take any object as its argument

How is string coercion implemented? October 10

Methods on Strings

Methods on Strings

Demo

Methods on Strings

```
>>> '1234'.isnumeric()
```

Demo

Methods on Strings

```
>>> '1234'.isnumeric()  
True
```

Demo

Methods on Strings

```
>>> '1234'.isnumeric()  
True
```

```
>>> 'r0BERT dE nIRO'.swapcase()
```

Demo

Methods on Strings

```
>>> '1234'.isnumeric()  
True
```

```
>>> 'r0BERT dE nIRO'.swapcase()  
'Robert De Niro'
```

Demo

Methods on Strings

```
>>> '1234'.isnumeric()  
True
```

```
>>> 'r0BERT dE nIRO'.swapcase()  
'Robert De Niro'
```

```
>>> 'snakeyes'.upper().endswith('YES')
```

Demo

Methods on Strings

```
>>> '1234'.isnumeric()  
True
```

```
>>> 'r0BERT dE nIRO'.swapcase()  
'Robert De Niro'
```

```
>>> 'snakeyes'.upper().endswith('YES')  
True
```

Demo

Sequences as Conventional Interfaces

Sequences as Conventional Interfaces

Consider two problems:

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

`enumerate naturals:`

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

`enumerate naturals: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.`

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

`enumerate naturals: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.`

`map fib:`

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

`enumerate naturals:` `1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.`

`map fib:` `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.`

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

`enumerate naturals:` `1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.`

`map fib:` `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.`

`filter iseven:`

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

`enumerate naturals:` 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

`map fib:` 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.



`filter iseven:`

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

`enumerate naturals:` 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

`map fib:` 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.



`filter iseven:` 0, 2, 8, 34, .

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

`enumerate naturals:` 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.

`map fib:` 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55.



`filter iseven:` 0, 2, 8, 34, .

`accumulate sum:`

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words:

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'

filter iscap:

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'



filter iscap:

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'
 ▲ ▲ ▲

filter iscap: 'University', 'California', 'Berkeley'

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'



filter iscap: 'University', 'California', 'Berkeley'

map first:

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'



filter iscap: 'University', 'California', 'Berkeley'

map first: 'U', 'C', 'B'

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first `n` Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'



filter iscap: 'University', 'California', 'Berkeley'

map first: 'U', 'C', 'B'

accumulate tuple:

Sequences as Conventional Interfaces

Consider two problems:

- Sum the even members of the first n Fibonacci numbers.
- List the letters in the acronym for a name, which includes the first letter of each capitalized word.

enumerate words: 'University', 'of', 'California', 'Berkeley'



filter iscap: 'University', 'California', 'Berkeley'

map first: 'U', 'C', 'B'

accumulate tuple: ('U', 'C', 'B')

Mapping a Function over a Sequence

Apply a function to each element of the sequence

Mapping a Function over a Sequence

Apply a function to each element of the sequence

```
>>> alternates = (-1, 2, -3, 4, -5)
```


Mapping a Function over a Sequence

Apply a function to each element of the sequence

```
>>> alternates = (-1, 2, -3, 4, -5)
```

```
>>> tuple(map(abs, alternates))
```

Mapping a Function over a Sequence

Apply a function to each element of the sequence

```
>>> alternates = (-1, 2, -3, 4, -5)
```

```
>>> tuple(map(abs, alternates))  
(1, 2, 3, 4, 5)
```

Mapping a Function over a Sequence

Apply a function to each element of the sequence

```
>>> alternates = (-1, 2, -3, 4, -5)
```

```
>>> tuple(map(abs, alternates))  
(1, 2, 3, 4, 5)
```

The returned value of map is an iterable map object

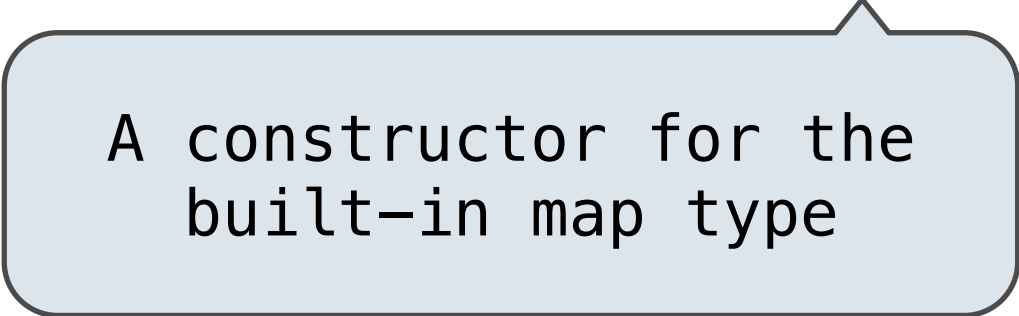
Mapping a Function over a Sequence

Apply a function to each element of the sequence

```
>>> alternates = (-1, 2, -3, 4, -5)
```

```
>>> tuple(map(abs, alternates))  
(1, 2, 3, 4, 5)
```

The returned value of `map` is an iterable map object



A constructor for the
built-in map type

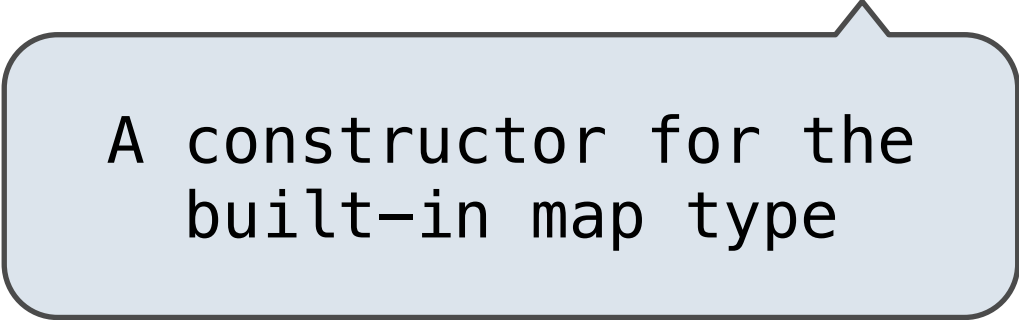
Mapping a Function over a Sequence

Apply a function to each element of the sequence

```
>>> alternates = (-1, 2, -3, 4, -5)
```

```
>>> tuple(map(abs, alternates))  
(1, 2, 3, 4, 5)
```

The returned value of `map` is an iterable map object



A constructor for the
built-in map type

The returned value of `filter` is an iterable filter object

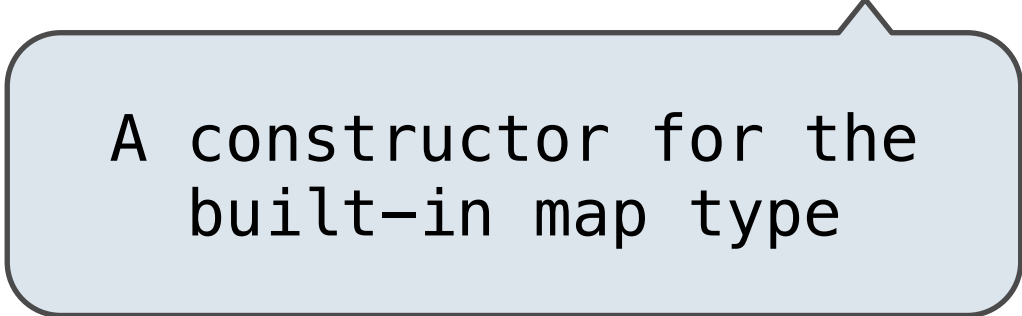
Mapping a Function over a Sequence

Apply a function to each element of the sequence

```
>>> alternates = (-1, 2, -3, 4, -5)
```

```
>>> tuple(map(abs, alternates))  
(1, 2, 3, 4, 5)
```

The returned value of `map` is an iterable map object



A constructor for the
built-in map type

The returned value of `filter` is an iterable filter object

Demo

Accumulation and Iterable Values

Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Many built-in functions take iterable objects as argument.

Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Many built-in functions take iterable objects as argument.

`tuple` Return a tuple containing the elements

Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Many built-in functions take iterable objects as argument.

`tuple` Return a tuple containing the elements

`sum` Return the sum of the elements

Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Many built-in functions take iterable objects as argument.

`tuple` Return a tuple containing the elements

`sum` Return the sum of the elements

`min` Return the minimum of the elements

Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Many built-in functions take iterable objects as argument.

`tuple` Return a tuple containing the elements

`sum` Return the sum of the elements

`min` Return the minimum of the elements

`max` Return the maximum of the elements

Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Many built-in functions take iterable objects as argument.

`tuple` Return a tuple containing the elements

`sum` Return the sum of the elements

`min` Return the minimum of the elements

`max` Return the maximum of the elements

For statements also operate on iterable values.

Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Many built-in functions take iterable objects as argument.

`tuple` Return a tuple containing the elements

`sum` Return the sum of the elements

`min` Return the minimum of the elements

`max` Return the maximum of the elements

For statements also operate on iterable values.

Demo

Generator Expressions

One large expression that evaluates to an iterable object

Generator Expressions

One large expression that evaluates to an iterable object

```
(<map exp> for <name> in <iter exp> if <filter exp>)
```

Generator Expressions

One large expression that evaluates to an iterable object

```
(<map exp> for <name> in <iter exp> if <filter exp>)
```

- Evaluates to an iterable object.

Generator Expressions

One large expression that evaluates to an iterable object

```
(<map exp> for <name> in <iter exp> if <filter exp>)
```

- Evaluates to an iterable object.
- `<iter exp>` is evaluated when the generator expression is evaluated.

Generator Expressions

One large expression that evaluates to an iterable object

```
(<map exp> for <name> in <iter exp> if <filter exp>)
```

- Evaluates to an iterable object.
- `<iter exp>` is evaluated when the generator expression is evaluated.
- Remaining expressions are evaluated when elements are accessed

Generator Expressions

One large expression that evaluates to an iterable object

```
(<map exp> for <name> in <iter exp> if <filter exp>)
```

- Evaluates to an iterable object.
- `<iter exp>` is evaluated when the generator expression is evaluated.
- Remaining expressions are evaluated when elements are accessed

```
(<map exp> for <name> in <iter exp>)
```

Generator Expressions

One large expression that evaluates to an iterable object

```
(<map exp> for <name> in <iter exp> if <filter exp>)
```

- Evaluates to an iterable object.
- `<iter exp>` is evaluated when the generator expression is evaluated.
- Remaining expressions are evaluated when elements are accessed

```
(<map exp> for <name> in <iter exp>)
```

Precise evaluation rule introduced in Chapter 4.

Reducing a Sequence

Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```
>>> from operator import mul
```

Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```
>>> from operator import mul
```

```
>>> from functools import reduce
```

Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```
>>> from operator import mul
>>> from functools import reduce
>>> reduce(mul, (1, 2, 3, 4, 5))
```

Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```
>>> from operator import mul
>>> from functools import reduce
>>> reduce(mul, (1, 2, 3, 4, 5))
120
```

Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```
>>> from operator import mul
>>> from functools import reduce
>>> reduce(mul, (1, 2, 3, 4, 5))
120
```

Similar to accumulate from Homework 2

Reducing a Sequence

Reduce is a higher-order generalization of max, min, & sum.

```
>>> from operator import mul
>>> from functools import reduce
>>> reduce(mul, (1, 2, 3, 4, 5))
120
```

Similar to accumulate from Homework 2

Demo