61A Lecture 13

Wednesday, October 2

Announcements

Announcements

- Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.

Announcements

- Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.
- Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.

Announcements

- Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.
- Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.
- Homework 4 due Tuesday 10/8 @ 11:59pm.

Announcements

- Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.
- Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.
- Homework 4 due Tuesday 10/8 @ 11:59pm.
- Project 2 due Thursday 10/10 @ 11:59pm.

Announcements

- Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.
- Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.
- Homework 4 due Tuesday 10/8 @ 11:59pm.
- Project 2 due Thursday 10/10 @ 11:59pm.
- Guerrilla Section 2 this Saturday 10/5 & Sunday 10/6 10am-1pm in Soda.

Announcements

- Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.
- Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.
- Homework 4 due Tuesday 10/8 @ 11:59pm.
- Project 2 due Thursday 10/10 @ 11:59pm.
- Guerrilla Section 2 this Saturday 10/5 & Sunday 10/6 10am-1pm in Soda.
  - Topics: Data abstraction, sequences, and non-local assignment.

Announcements

- Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.
- Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.
- Homework 4 due Tuesday 10/8 @ 11:59pm.
- Project 2 due Thursday 10/10 @ 11:59pm.
- Guerrilla Section 2 this Saturday 10/5 & Sunday 10/6 10am–1pm in Soda.
  - Topics: Data abstraction, sequences, and non-local assignment.
  - Please RSVP on Piazza!

Announcements

- Homework 3 deadline extended to Wednesday 10/2 @ 11:59pm.
- Optional Hog strategy contest due Thursday 10/3 @ 11:59pm.
- Homework 4 due Tuesday 10/8 @ 11:59pm.
- Project 2 due Thursday 10/10 @ 11:59pm.
- Guerrilla Section 2 this Saturday 10/5 & Sunday 10/6 10am–1pm in Soda.
  - Topics: Data abstraction, sequences, and non-local assignment.
  - Please RSVP on Piazza!
- Guest lecture on Wednesday 10/9, Peter Norvig on Natural Language Processing in Python.

Strings

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:

```
"""And, as imagination bodies forth
The forms of things to unknown, and the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name.
"""
```

Strings are an Abstraction

Representing data:

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

Representing language:

```
"""And, as imagination bodies forth
The forms of things to unknown, and the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name.
"""
```

Representing programs:

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

Strings are an Abstraction

Representing data:

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

Representing language:

```
"""And, as imagination bodies forth
The forms of things to unknown, and the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name.
"""
```

Representing programs:

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

(Demo)

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-quoted 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-quoted 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-quoted 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-quoted and double-quoted strings are equivalent

A backslash "escapes" the following character

"Line feed" character represents a new line

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, but with only one character!

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, but with only one character!

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.

(Demo)

String Membership Differs from Other Sequence Types

String Membership Differs from Other Sequence Types

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

String Membership Differs from Other Sequence Types

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

```
>>> 'here' in "Where's Waldo?"
True
>>> 234 in (1, 2, 3, 4, 5)
False
```

String Membership Differs from Other Sequence Types

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

```
>>> 'here' in "Where's Waldo?"
True
>>> 234 in (1, 2, 3, 4, 5)
False
```

Why? Working with strings, we usually care about words more than characters

String Membership Differs from Other Sequence Types

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

```
>>> 'here' in "Where's Waldo?"
True
>>> 234 in (1, 2, 3, 4, 5)
False
```

Why? Working with strings, we usually care about words more than characters

The count method also matches substrings

String Membership Differs from Other Sequence Types

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

```
>>> 'here' in "Where's Waldo?"
True
>>> 234 in (1, 2, 3, 4, 5)
False
```

Why? Working with strings, we usually care about words more than characters

The count method also matches substrings

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

String Membership Differs from Other Sequence Types

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

```
>>> 'here' in "Where's Waldo?"
True
>>> 234 in (1, 2, 3, 4, 5)
False
```

Why? Working with strings, we usually care about words more than 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 Sequence Types

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

```
>>> 'here' in "Where's Waldo?"
True
>>> 234 in (1, 2, 3, 4, 5)
False
```

Why? Working with strings, we usually care about words more than characters

The count method also matches substrings

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

the number of non-overlapping occurrences of a substring

Encoding Strings

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

8 rows: 3 bits

	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
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

8 rows: 3 bits

16 columns: 4 bits

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange

ASCII Code Chart

8 rows: 3 bits	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

16 columns: 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

8 rows: 3 bits	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

16 columns: 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

8 rows: 3 bits

	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

16 columns: 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

"Line feed" (\n)

ASCII Code Chart

8 rows: 3 bits

	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

16 columns: 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

"Bell" (\a)

"Line feed" (\n)

ASCII Code Chart

8 rows: 3 bits

	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

16 columns: 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

"Bell" (\a)

"Line feed" (\n)

ASCII Code Chart

8 rows: 3 bits

	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

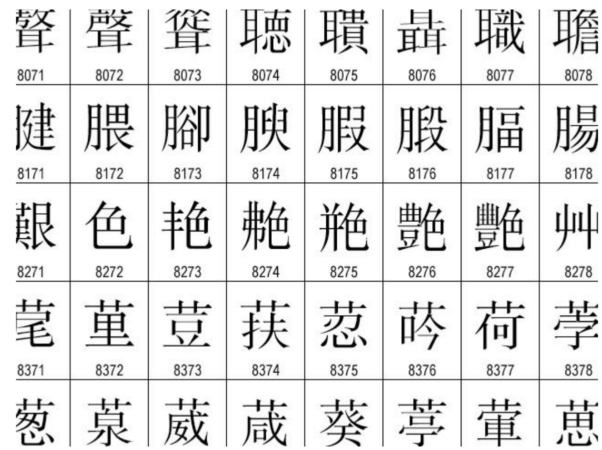
16 columns: 4 bits

- 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

(Demo)

Representing Strings: the Unicode Standard

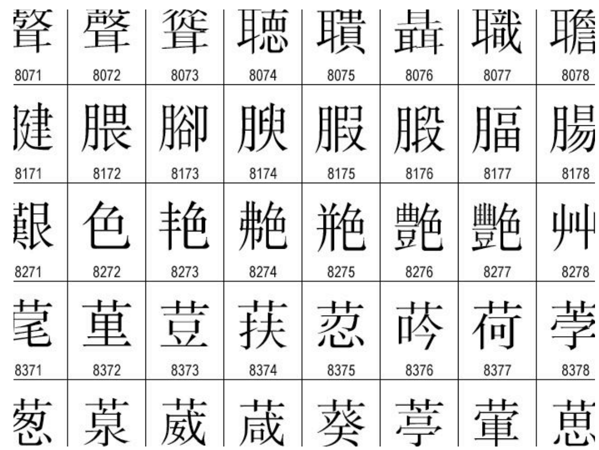
Representing Strings: the Unicode Standard



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

Representing Strings: the Unicode Standard

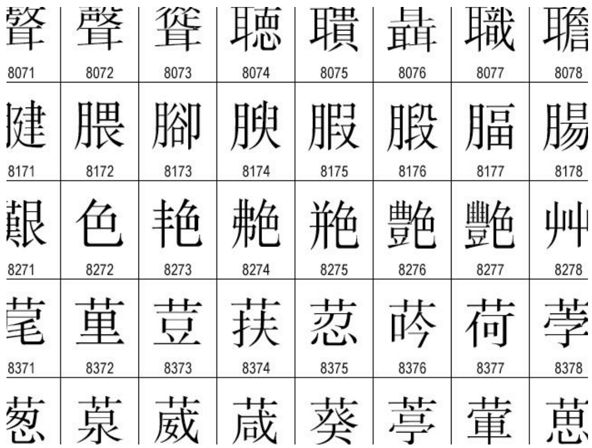
- 109,000 characters



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

Representing Strings: the Unicode Standard

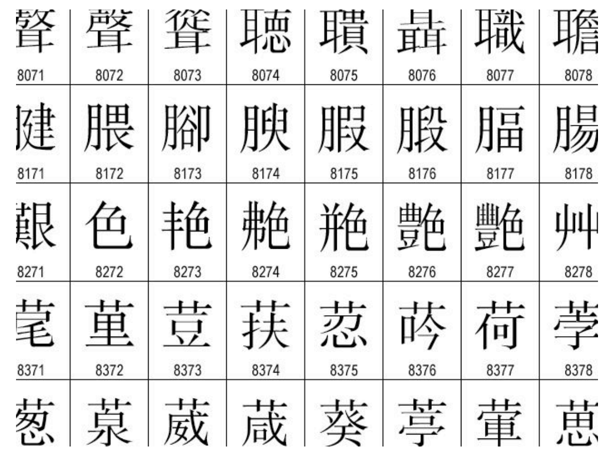
- 109,000 characters
- 93 scripts (organized)



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

Representing Strings: the Unicode Standard

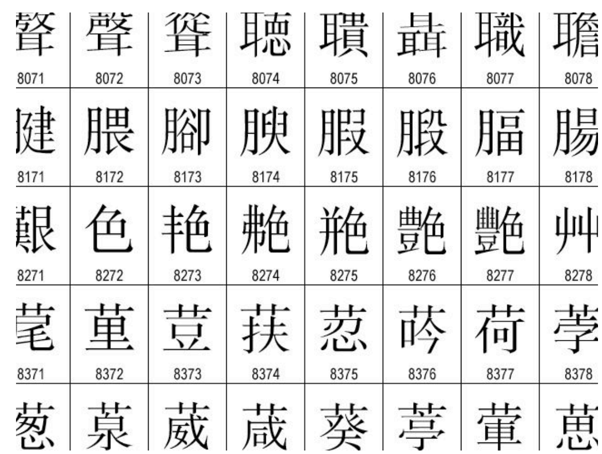
- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case



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

Representing Strings: the Unicode Standard

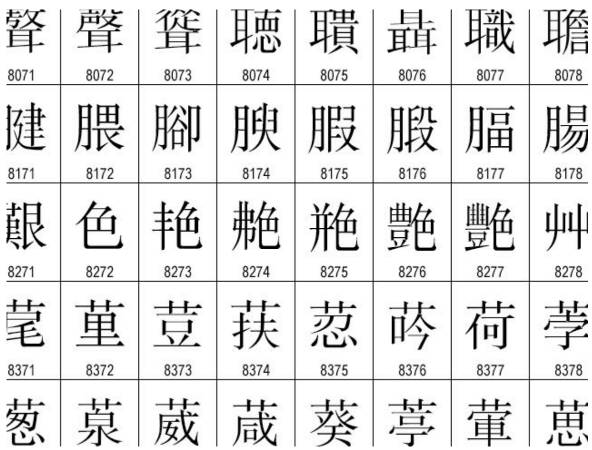
- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order



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

Representing Strings: the Unicode Standard

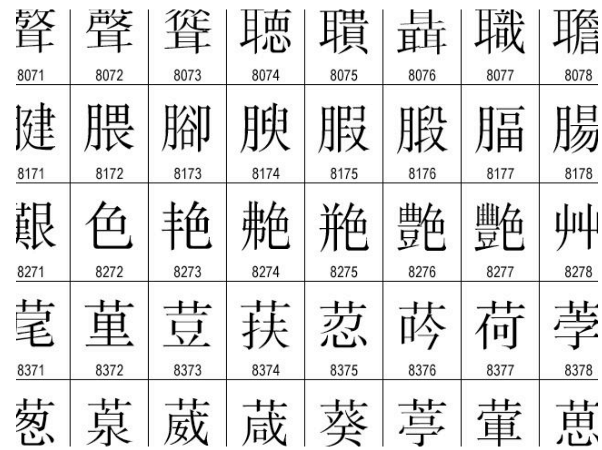
- 109,000 characters
- 93 scripts (organized)
- Enumeration of character properties, such as case
- Supports bidirectional display order
- A canonical name for every character



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
- A canonical name for every character

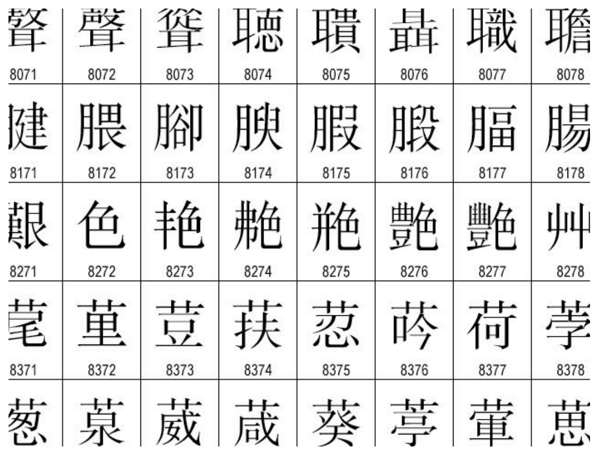


http://ian-albert.com/unicode_chart/unichart-chinese.jpg

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
- A canonical name for every character



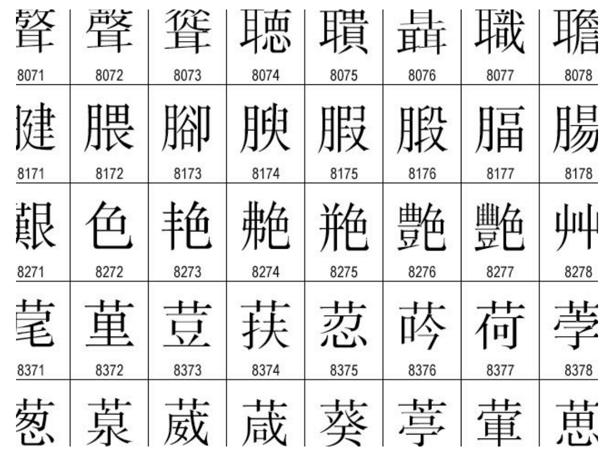
http://ian-albert.com/unicode_chart/unichart-chinese.jpg

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
- A canonical name for every character



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

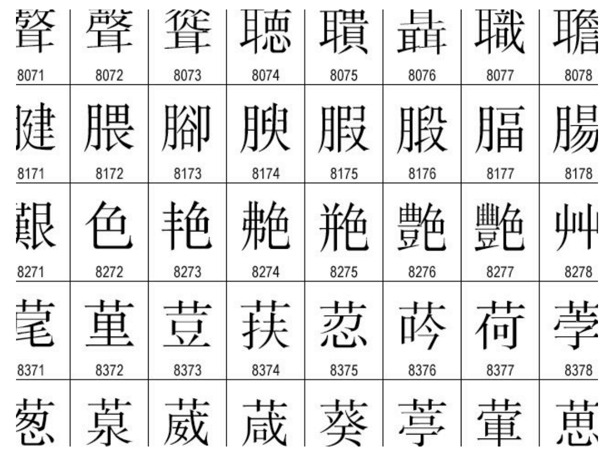
U+0058 LATIN CAPITAL LETTER X

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
- A canonical name for every character



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

U+0058 LATIN CAPITAL LETTER X

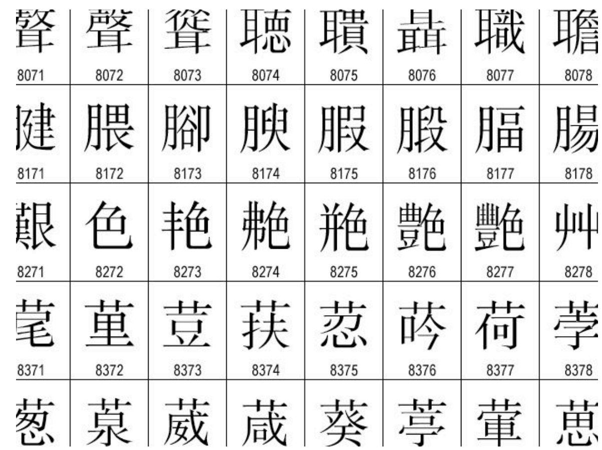
U+263a WHITE SMILING FACE

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
- A canonical name for every character



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

U+0058 LATIN CAPITAL LETTER X

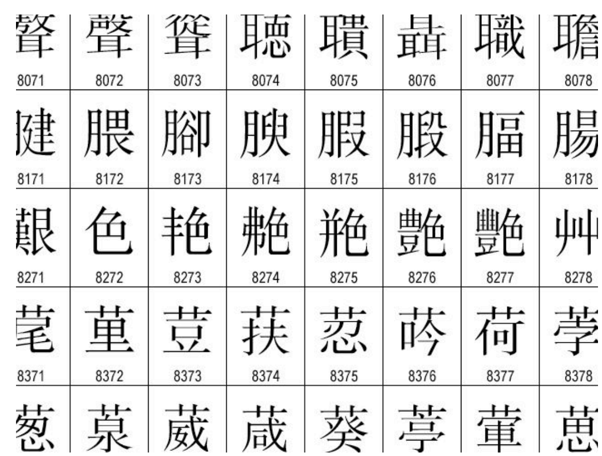
U+263a WHITE SMILING FACE

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
- A canonical name for every character



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

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 integers

Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between those integers and bytes

Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between those integers 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 integers

UTF-8: Correspondence between those integers 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 integers

UTF-8: Correspondence between those integers 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 integers

UTF-8: Correspondence between those integers 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 integers

UTF-8: Correspondence between those integers and bytes

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

bytes			integers
	00000000	0	
	00000001	1	

Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between those integers and bytes

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

bytes	00000000	0	integers
	00000001	1	
	00000010	2	

Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between those integers and bytes

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

bytes			integers
	00000000	0	
	00000001	1	
	00000010	2	
	00000011	3	

Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between those integers and bytes

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

bytes			integers
	00000000	0	
	00000001	1	
	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 integers

UTF-8: Correspondence between those integers and bytes

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

bytes			integers
	00000000	0	
	00000001	1	
	00000010	2	
	00000011	3	

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

In Python: string length is measured in characters, bytes length in bytes.

Representing Strings: UTF-8 Encoding

UTF (UCS (Universal Character Set) Transformation Format)

Unicode: Correspondence between characters and integers

UTF-8: Correspondence between those integers and bytes

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

bytes		integers
00000000	0	
00000001	1	
00000010	2	
00000011	3	

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

In Python: string length is measured in characters, bytes length in bytes.

(Demo)

Sequence Processing

Sequence Processing

Sequence Processing

```
Consider two problems:
```

Sequence Processing

Consider two problems:

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

Sequence Processing

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.

Sequence Processing

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.

Sequence Processing

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:

Sequence Processing

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.

Sequence Processing

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:
```

Sequence Processing

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.

Sequence Processing

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 even:	

Sequence Processing

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 even:	

Sequence Processing

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 even:	0, 2, 8, 34, .

Sequence Processing

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 even:            0,       2,       8,           34,    .

accumulate sum:
```

Sequence Processing

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 even:           0,       2,       8,           34,    .

accumulate sum:        .,       .,       .,           .,  =44
```

Sequence Processing

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.

Sequence Processing

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.

Sequence Processing

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:

Sequence Processing

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'

Sequence Processing

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 capitalized:

Sequence Processing

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 capitalized:

Sequence Processing

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 capitalized:	'University',		'California',	'Berkeley'

Sequence Processing

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 capitalized: 'University', 'California', 'Berkeley'

map first:

Sequence Processing

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 capitalized:	'University',		'California',	'Berkeley'
map first:	'U',		'C',	'B'

Sequence Processing

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 capitalized:	'University',		'California',	'Berkeley'
map first:	'U',		'C',	'B'
accumulate tuple:				

Sequence Processing

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 capitalized:		'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)

Iteration and Accumulation

Iterable Values and Accumulation

Iterable Values and Accumulation

Iterable objects give access to their elements in order.

Iterable Values and Accumulation

Iterable objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

Iterable Values and Accumulation

Iterable objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

Many built-in functions take iterable objects as argument.

Iterable Values and Accumulation

Iterable objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

Many built-in functions take iterable objects as argument.

tuple	Return a tuple containing the elements

Iterable Values and Accumulation

Iterable objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

Many built-in functions take iterable objects as argument.

tuple	Return a tuple containing the elements
sum	Return the sum of the elements

Iterable Values and Accumulation

Iterable objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

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

Iterable Values and Accumulation

Iterable objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

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

Iterable Values and Accumulation

Iterable objects give access to their elements in order.

Similar to a sequence, but does not always allow element selection or have finite length.

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.

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
```

First argument:
A two-argument function

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
```

First argument:
A two-argument function

Second argument: an
iterable object

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
```

First argument: A two-argument function

Second argument: an iterable object

Similar to accumulate from Homework 2, but with iterable objects.

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.

Short version: (<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.

Short version: (<map exp> for <name> in <iter exp>)

(Demo)