

61A Lecture 12

Announcements

Objects

(Demo)

Objects

Objects

- Objects represent information

Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions

Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes

Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python

Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:

Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
 - A metaphor for organizing large programs

Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
 - A metaphor for organizing large programs
 - Special syntax that can improve the composition of programs

Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
 - A metaphor for organizing large programs
 - Special syntax that can improve the composition of programs
- In Python, every value is an object

Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
 - A metaphor for organizing large programs
 - Special syntax that can improve the composition of programs
- In Python, every value is an object
 - All **objects** have **attributes**

Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
 - A metaphor for organizing large programs
 - Special syntax that can improve the composition of programs
- In Python, every value is an object
 - All **objects** have **attributes**
 - A lot of data manipulation happens through object **methods**

Objects

- Objects represent information
- They consist of data and behavior, bundled together to create abstractions
- Objects can represent things, but also properties, interactions, & processes
- A type of object is called a class; **classes** are first-class values in Python
- Object-oriented programming:
 - A metaphor for organizing large programs
 - Special syntax that can improve the composition of programs
- In Python, every value is an object
 - All **objects** have **attributes**
 - A lot of data manipulation happens through object **methods**
 - Functions do one thing; objects do many related things

Example: Strings

(Demo)

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
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															
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

- 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

8 rows: 3 bits

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

	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

- 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	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

"Line feed" (\n)

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

8 rows: 3 bits

16 columns: 4 bits

"Bell" (\a) "Line feed" (\n)

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

8 rows: 3 bits

16 columns: 4 bits

"Bell" (\a) "Line feed" (\n)

- 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

聳	聲	聳	聰	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	膾	腳	腴	暇	睂	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
耄	堇	荳	菴	葱	苓	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菸	葳	歲	葵	葶	葷	蕙

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

Representing Strings: the Unicode Standard

- 109,000 characters

聳	聲	聳	聽	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	膾	腳	腴	暇	睂	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
靉	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
耄	堇	荳	菝	葱	苓	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菴	葳	葳	葵	葶	葷	蕙

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

Representing Strings: the Unicode Standard

- 109,000 characters
- 93 scripts (organized)

聳	聲	聳	聽	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	膾	腳	腴	暇	睂	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
靉	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
耄	堇	荳	菴	葱	苓	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菸	葳	歲	葵	葶	葷	蕙

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

聳	聲	聳	聰	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	脻	腳	腴	暇	睂	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
耄	堇	荳	菝	葱	苓	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菴	葳	歲	葵	葶	葷	蕙

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

聳	聲	聳	聰	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	膾	腳	腴	暇	睂	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
毫	菴	荳	菴	葱	苓	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菴	葳	歲	葵	葶	葷	蕙

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

聳	聲	聳	聰	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	脕	腳	腴	暇	睂	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
耄	堇	荳	菴	葱	苓	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菸	葳	歲	葵	葶	葷	蕙

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

聳	聲	聳	聰	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	脻	腳	腴	暇	睂	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
耄	堇	荳	菝	葱	苓	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菴	葳	歲	葵	葶	葷	蕙

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

聳	聲	聳	聰	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	脻	腳	腴	暇	脜	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
芼	菑	荳	菴	葱	陔	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菑	歲	歲	葵	葶	葷	蕙

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

聳	聲	聳	聰	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	脻	腳	腴	暇	脜	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
毫	菴	荳	菴	葱	苓	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菴	葳	歲	葵	葶	葷	蕙

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

聳	聲	聳	聰	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	脻	腳	腴	暇	脜	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
芼	菑	荳	菴	葱	陔	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菑	歲	歲	葵	葶	葷	蕙

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

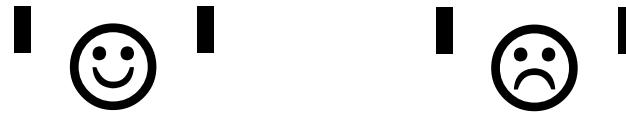
聳	聲	聳	聰	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	膾	腳	腴	暇	睂	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
毫	菴	荳	菴	葱	苓	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菴	葳	歲	葵	葶	葷	蕙

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

聳	聲	聳	聰	蹠	聶	職	瞻
8071	8072	8073	8074	8075	8076	8077	8078
健	脻	腳	腴	暇	脜	脢	腸
8171	8172	8173	8174	8175	8176	8177	8178
艱	色	艳	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
毫	菴	荳	菴	葱	苓	荷	荸
8371	8372	8373	8374	8375	8376	8377	8378
葱	菴	葳	歲	葵	葶	葷	蕙

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)

Mutation Operations

Some Objects Can Change

[Demo]

Some Objects Can Change

[Demo]

First example in the course of an object changing state

Some Objects Can Change

[Demo]

First example in the course of an object changing state

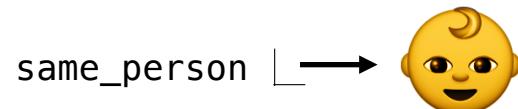
The same object can change in value throughout the course of computation

Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

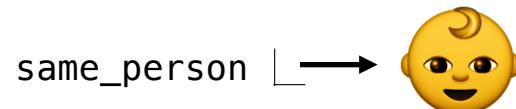


Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation



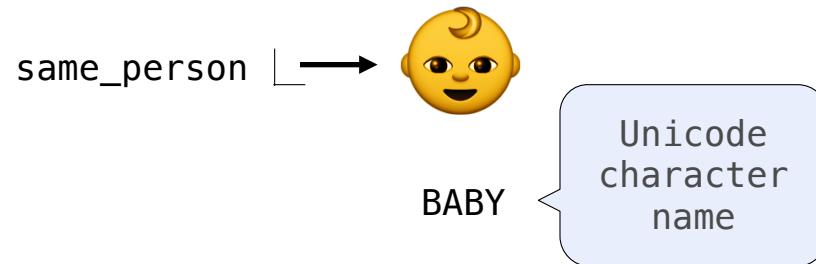
BABY

Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

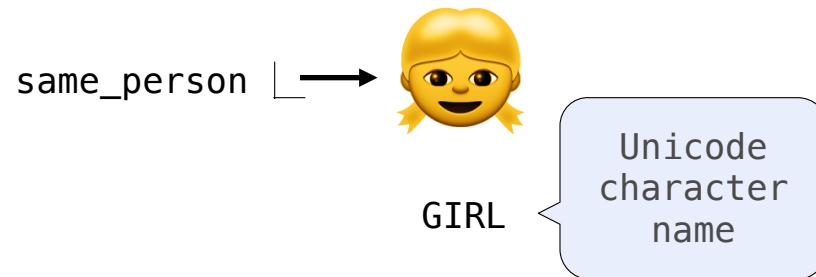


Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

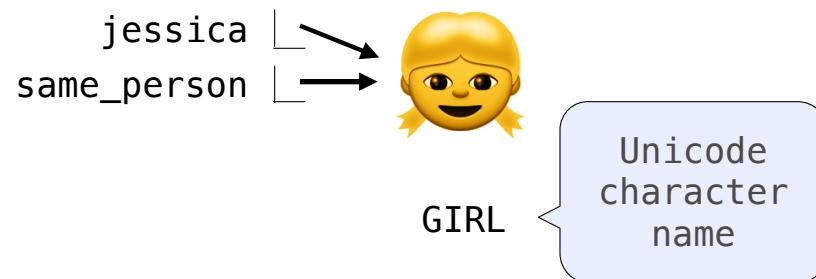


Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

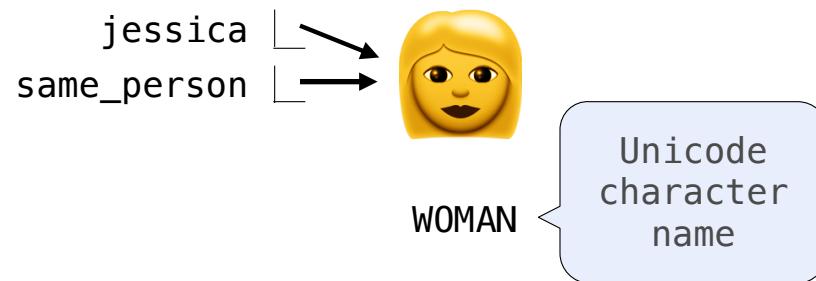


Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

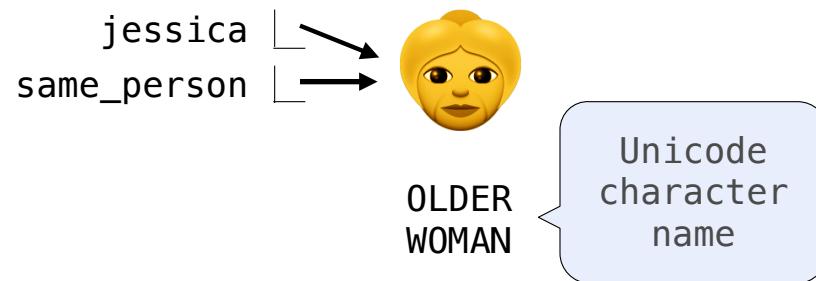


Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation

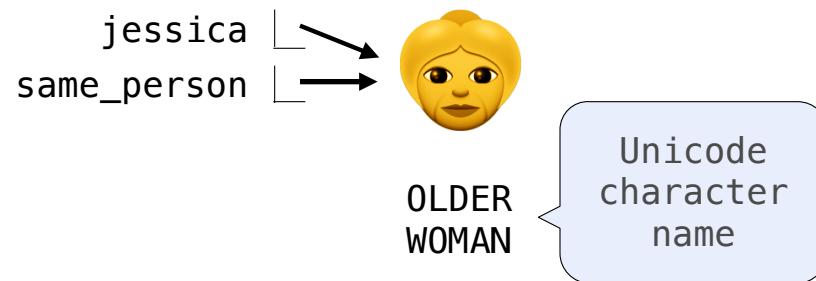


Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation



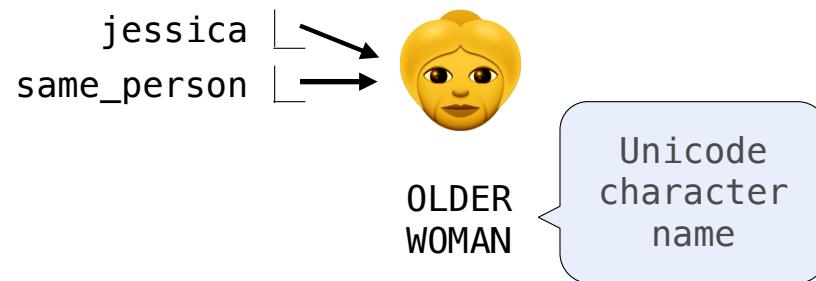
All names that refer to the same object are affected by a mutation

Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation



All names that refer to the same object are affected by a mutation

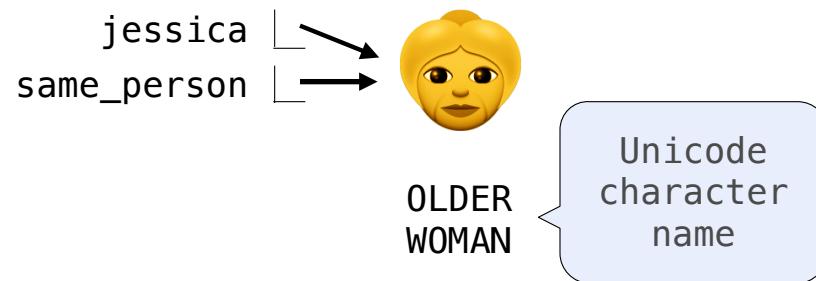
Only objects of *mutable* types can change: lists & dictionaries

Some Objects Can Change

[Demo]

First example in the course of an object changing state

The same object can change in value throughout the course of computation



All names that refer to the same object are affected by a mutation

Only objects of *mutable* types can change: lists & dictionaries

{Demo}

Limitations on Dictionaries

Limitations on Dictionaries

Dictionaries are **unordered** collections of key-value pairs

Limitations on Dictionaries

Dictionaries are **unordered** collections of key-value pairs

Dictionary keys do have two restrictions:

Limitations on Dictionaries

Dictionaries are **unordered** collections of key-value pairs

Dictionary keys do have two restrictions:

- A key of a dictionary **cannot be** a list or a dictionary (or any *mutable type*)

Limitations on Dictionaries

Dictionaries are **unordered** collections of key-value pairs

Dictionary keys do have two restrictions:

- A key of a dictionary **cannot be** a list or a dictionary (or any *mutable type*)
- Two **keys cannot be equal**; There can be at most one value for a given key

Limitations on Dictionaries

Dictionaries are **unordered** collections of key-value pairs

Dictionary keys do have two restrictions:

- A key of a dictionary **cannot be** a list or a dictionary (or any *mutable type*)
- Two **keys cannot be equal**; There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries

Limitations on Dictionaries

Dictionaries are **unordered** collections of key-value pairs

Dictionary keys do have two restrictions:

- A key of a dictionary **cannot be** a list or a dictionary (or any *mutable type*)
- Two **keys cannot be equal**; There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries

The second restriction is part of the dictionary abstraction

Limitations on Dictionaries

Dictionaries are **unordered** collections of key-value pairs

Dictionary keys do have two restrictions:

- A key of a dictionary **cannot be** a list or a dictionary (or any *mutable type*)
- Two **keys cannot be equal**; There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries

The second restriction is part of the dictionary abstraction

If you want to associate multiple values with a key, store them all in a sequence value:
`{'odds': [1, 3, 5], 'evens': [2, 4, 6]}`

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
```

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
```

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
```

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```

```
def mystery(s):
    s.pop()
    s.pop()
```

[Interactive Diagram](#)

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```

```
def mystery(s): or def mystery(s):
    s.pop()           s[2:] = []
```

[Interactive Diagram](#)

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2

>>> four = [1, 2, 3, 4]
```

```
def mystery(s): or def mystery(s):
    s.pop()           s[2:] = []
    s.pop()
```

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
```

```
def mystery(s): or def mystery(s):
    s.pop()           s[2:] = []
    s.pop()
```

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```

```
def mystery(s): or def mystery(s):
    s.pop()           s[2:] = []
```

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> another_mystery() # No arguments!
```

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```

```
def mystery(s): or def mystery(s):
    s.pop()           s[2:] = []
```

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> another_mystery() # No arguments!
>>> len(four)
2
```

[Interactive Diagram](#)

Mutation Can Happen Within a Function Call

A function can change the value of any object in its scope

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> mystery(four)
>>> len(four)
2
```

```
def mystery(s): or def mystery(s):
    s.pop()           s[2:] = []
    s.pop()
```

```
>>> four = [1, 2, 3, 4]
>>> len(four)
4
>>> another_mystery() # No arguments!
>>> len(four)
2
```

```
def another_mystery():
    four.pop()
    four.pop()
```

[Interactive Diagram](#)

Tuples

(Demo)

Tuples are Immutable Sequences

Tuples are Immutable Sequences

Immutable values are protected from mutation

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)  
>>> ooze()  
>>> turtle  
(1, 2, 3)
```

```
>>> turtle = [1, 2, 3]
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)  
>>> ooze()  
>>> turtle  
(1, 2, 3)
```

```
>>> turtle = [1, 2, 3]  
>>> ooze()
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)  
>>> ooze()  
>>> turtle  
(1, 2, 3)
```

```
>>> turtle = [1, 2, 3]  
>>> ooze()  
>>> turtle
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

```
>>> x + x
```

Name change:

```
>>> x + x
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

```
>>> x = 2
>>> x + x
```

Name change:

```
>>> x + x
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x + x
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x + x
>>> x + x
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
>>> x + x
[1, 2, 1, 2]
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x + x
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it *contains* a mutable value as an element

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it *contains* a mutable value as an element

```
>>> s = ([1, 2], 3)
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it *contains* a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it *contains* a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it *contains* a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```

```
>>> s = ([1, 2], 3)
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it *contains* a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```

```
>>> s = ([1, 2], 3)
>>> s[0][0] = 4
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it *contains* a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```

```
>>> s = ([1, 2], 3)
>>> s[0][0] = 4
>>> s
```

Tuples are Immutable Sequences

Immutable values are protected from mutation

```
>>> turtle = (1, 2, 3)
>>> ooze()
>>> turtle
(1, 2, 3)
```

Next lecture: ooze can change turtle's binding

```
>>> turtle = [1, 2, 3]
>>> ooze()
>>> turtle
['Anything could be inside!']
```

The value of an expression can change because of changes in names or objects

Name change:

```
>>> x = 2
>>> x + x
4
>>> x = 3
>>> x + x
6
```

Object mutation:

```
>>> x = [1, 2]
>>> x + x
[1, 2, 1, 2]
>>> x.append(3)
>>> x + x
[1, 2, 3, 1, 2, 3]
```

An immutable sequence may still change if it *contains* a mutable value as an element

```
>>> s = ([1, 2], 3)
>>> s[0] = 4
ERROR
```

```
>>> s = ([1, 2], 3)
>>> s[0][0] = 4
>>> s
([4, 2], 3)
```

Mutation

Sameness and Change

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```
>>> a = [10]
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```
>>> a = [10]
>>> b = a
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```
>>> a = [10]
>>> b = a
>>> a == b
True
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> a
[10, 20]
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> a
[10, 20]
>>> b
[10, 20]
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> a
[10, 20]
>>> b
[10, 20]
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]                                >>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> a
[10, 20]
>>> b
[10, 20]
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> a
[10, 20]
>>> b
[10, 20]
```

```
>>> a = [10]
>>> b = [10]
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> a
[10, 20]
>>> b
[10, 20]
```

```
>>> a = [10]
>>> b = [10]
>>> a == b
True
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> a
[10, 20]
>>> b
[10, 20]
```

```
>>> a = [10]
>>> b = [10]
>>> a == b
True
>>> b.append(20)
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> a
[10, 20]
>>> b
[10, 20]
```

```
>>> a = [10]
>>> b = [10]
>>> a == b
True
>>> b.append(20)
>>> a
[10]
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> a
[10, 20]
>>> b
[10, 20]
```

```
>>> a = [10]
>>> b = [10]
>>> a == b
True
>>> b.append(20)
>>> a
[10]
>>> b
[10, 20]
```

Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- A rational number is just its numerator and denominator
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a == b
True
>>> a
[10, 20]
>>> b
[10, 20]
```

```
>>> a = [10]
>>> b = [10]
>>> a == b
True
>>> b.append(20)
>>> a
[10]
>>> b
[10, 20]
>>> a == b
False
```

Identity Operators

Identity Operators

Identity

`<exp0> is <exp1>`

evaluates to True if both `<exp0>` and `<exp1>` evaluate to the same object

Identity Operators

Identity

`<exp0> is <exp1>`

evaluates to `True` if both `<exp0>` and `<exp1>` evaluate to the same object

Equality

`<exp0> == <exp1>`

evaluates to `True` if both `<exp0>` and `<exp1>` evaluate to equal values

Identity Operators

Identity

`<exp0> is <exp1>`

evaluates to `True` if both `<exp0>` and `<exp1>` evaluate to the same object

Equality

`<exp0> == <exp1>`

evaluates to `True` if both `<exp0>` and `<exp1>` evaluate to equal values

Identical objects are always equal values

Identity Operators

Identity

`<exp0> is <exp1>`

evaluates to `True` if both `<exp0>` and `<exp1>` evaluate to the same object

Equality

`<exp0> == <exp1>`

evaluates to `True` if both `<exp0>` and `<exp1>` evaluate to equal values

Identical objects are always equal values

(Demo)

Mutable Default Arguments are Dangerous

Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):  
...     s.append(3)  
...     return len(s)  
...
```

Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
```

Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
>>> f()
2
```

Mutable Default Arguments are Dangerous

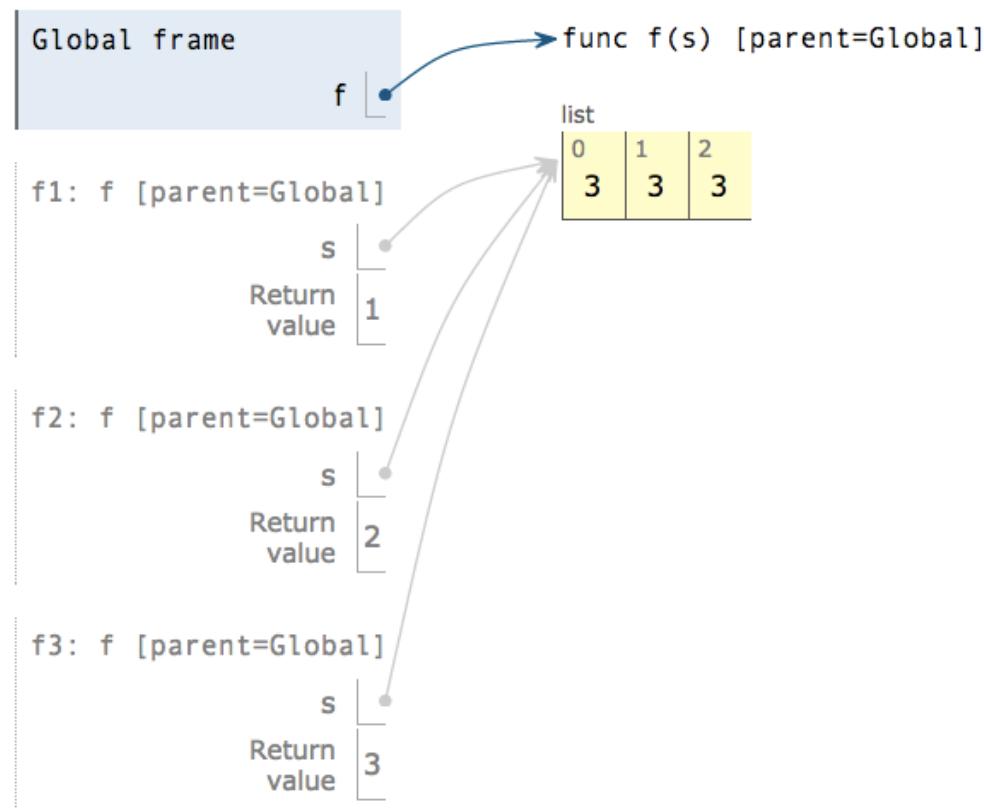
A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):
...     s.append(3)
...     return len(s)
...
>>> f()
1
>>> f()
2
>>> f()
3
```

Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):  
...     s.append(3)  
...     return len(s)  
...  
>>> f()  
1  
>>> f()  
2  
>>> f()  
3
```

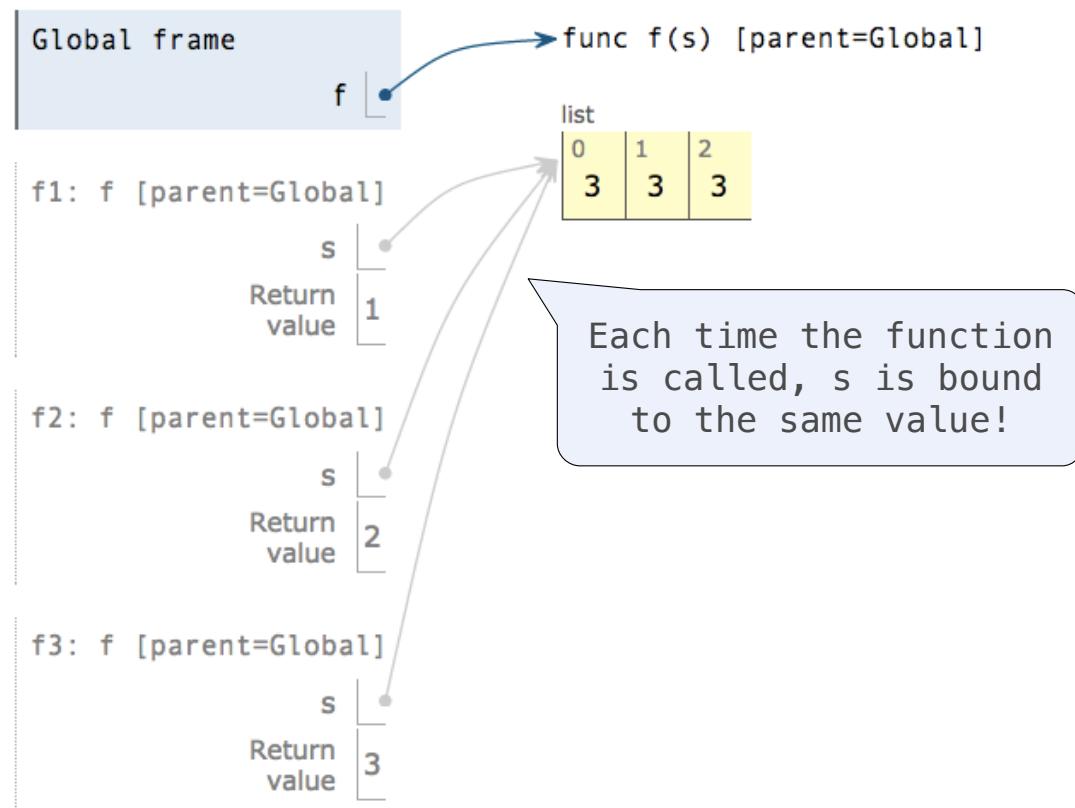


[Interactive Diagram](#)

Mutable Default Arguments are Dangerous

A default argument value is part of a function value, not generated by a call

```
>>> def f(s=[]):  
...     s.append(3)  
...     return len(s)  
...  
>>> f()  
1  
>>> f()  
2  
>>> f()  
3
```



[Interactive Diagram](#)