

Lecture 15.Mutable values

Objects

an object is suppose to behave like the values it has.

```
>>> from datetime import date
>>> date
<class 'datetime.date'>
>>> today = date(2015, 2, 20)
>>> today
datetime.date(2015, 2, 20)
>>> freedom = date(2015, 5, 12)
>>> str(freedom - today)
'81 days, 0:00:00'
>>> today.year
2015
>>> today.month
2
>>>
```

deeper introduction of th

当我们在Python中更

in addition to values bounded to the objects, there are also functions bounded to the objects, which are called method.

```
today.strftime('%A %B %d')
```

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

String

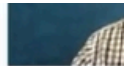
```
>>> s = 'Hello'
>>> s.upper()
'HELLO'
>>> s.lower()
'hello'
>>> s.swapcase()
'hELLO'
>>> s
'Hello'
>>> █
```



I

Representing Strings: the ASCII Standard

American Standard Code for Information Interchange



"Bell" (\a)

ASCII Code Chart

"Line feed" (\n)

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 designed for teletype and these things still exist today
- Rows indexed 2-5 are a useful 6-bit (64 element) subset
- Control characters were designed for transmission

Representing Strings: the Unicode Standard

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

聾	聾	聾	聾	聾	聾	聾	聾
8071	8072	8073	8074	8075	8076	8077	8078
健	腓	脚	腓	腓	腓	腓	腓
8171	8172	8173	8174	8175	8176	8177	8178
艷	色	艷	艷	艷	艷	艷	艷
8271	8272	8273	8274	8275	8276	8277	8278
菰	菰	荳	荳	荳	荳	荳	荳
8371	8372	8373	8374	8375	8376	8377	8378
葱	菰	菰	菰	菰	菰	菰	菰

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)

and most programming languages support unicode

```
>>> lookup('SNOWMAN')
```



```
>>> lookup('SOCCER BALL')
```



```
>>> lookup('BABY')
```



```
>>> 
```

Mutation Operations

Some objects can change

```

Python Python lec — Python — 104x27
>>> suits = ['coin', 'string', 'myriad']
>>> original_suits = suits
>>> suits.pop()
'myriad'
>>> suits.remove('string')
>>> suits
['coin']
>>> suits.append('cup')
>>> suits.extend(['sword', 'club'])
>>> suits
['coin', 'cup', 'sword', 'club']
>>> suits[2] = 'spade'
>>> suits[0:2] = ['heart', 'diamond']
>>> suits
['heart', 'diamond', 'spade', 'club']
>>> original_suits
['heart', 'diamond', 'spade', 'club']
>>>

```

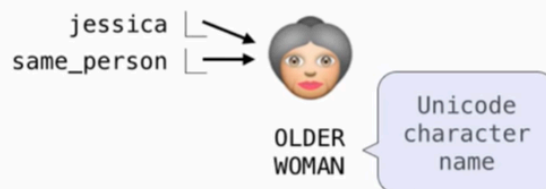


Some Objects Can Change

[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

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()

```



Immutable values

Tuples

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
```

Mutations

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]
```

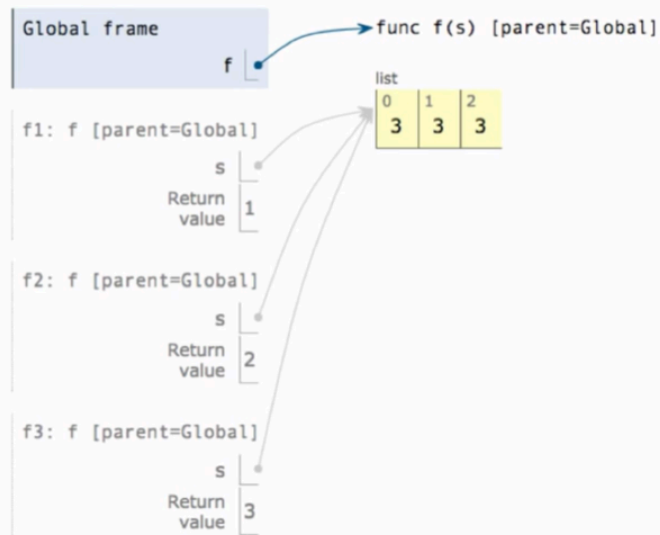
```
>>> a = [10]
>>> b = [10]
>>> a == b
True
>>> b.append(20)
>>> a
[10]
>>> b
[10, 20]
>>> a == b
False
```

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(5)
...     return len(s)
...
>>> f()
1
>>> f()
2
>>> f()
3
```



Interactive Diagram

简单的来说，在python中，tuple，int这些immutable value在赋值的时候只能整体赋值，当重新赋值的时候，事实上他们是指向不同的值（name bound to values）而不是相同的地址。而mutable values，比如list，在创建的时候，比如[]，就已经给定了这一个位置，name bound的是位置，而不是值。

Lists in Environment Diagrams

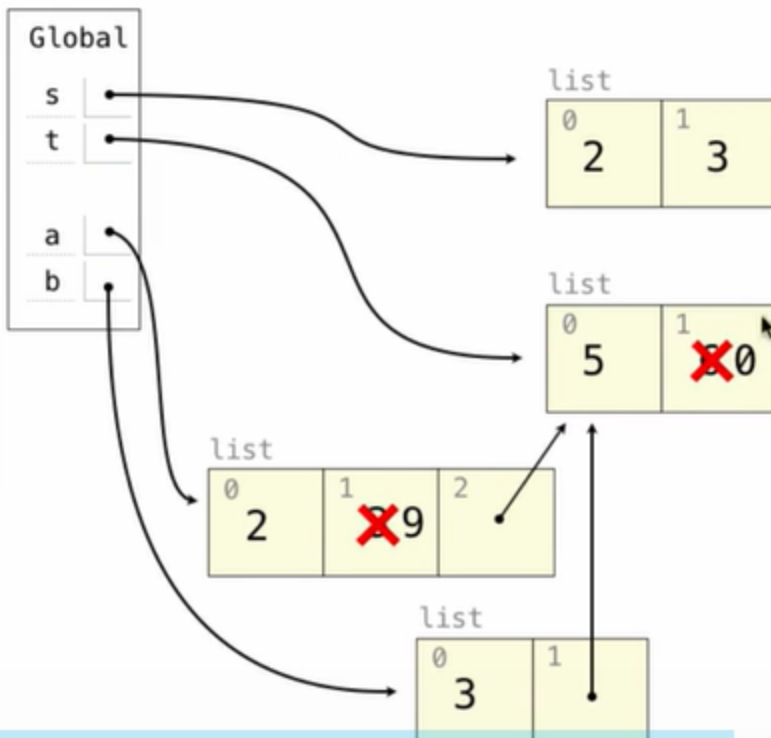
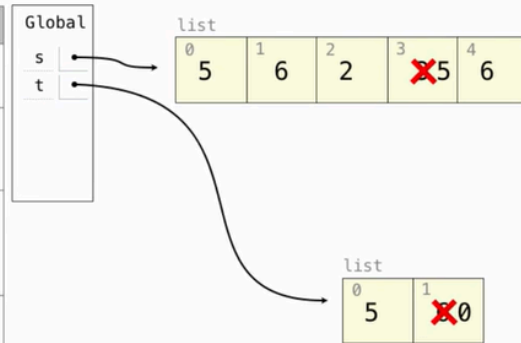


Assume that before each example below we execute:

`s = [2, 3]`

`t = [5, 6]`

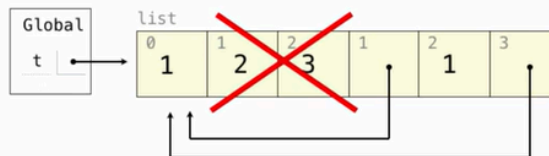
Operation	Example	Result
append adds one element to a list	<code>s.append(t)</code> <code>t = 0</code>	<code>s</code> → [2, 3, [5, 6]] <code>t</code> → 0
extend adds all elements in one list to another list	<code>s.extend(t)</code> <code>t[1] = 0</code>	<code>s</code> → [2, 3, 5, 6] <code>t</code> → [5, 0]
addition & slicing create new lists containing existing elements	<code>a = s + [t]</code> <code>b = a[1:]</code> <code>a[1] = 9</code> <code>b[1][1] = 0</code>	<code>s</code> → [2, 3] <code>t</code> → [5, 0] <code>a</code> → [2, 9, [5, 0]] <code>b</code> → [3, [5, 0]]
The list function also creates a new list containing existing elements	<code>t = list(s)</code> <code>s[1] = 0</code>	<code>s</code> → [2, 0] <code>t</code> → [2, 3]
slice assignment replaces a slice with new values	<code>s[0:0] = t</code> <code>s[3:] = t</code> <code>t[1] = 0</code>	<code>s</code> → [5, 6, 2, 5, 6] <code>t</code> → [5, 0]



Operation	Example	Result
pop removes & returns the last element	<code>t = s.pop()</code>	<code>s → [2]</code> <code>t → 3</code>
remove removes the first element equal to the argument	<code>t.extend(t)</code> <code>t.remove(5)</code>	<code>s → [2, 3]</code> <code>t → [6, 5, 6]</code>
slice assignment can remove elements from a list by assigning <code>[]</code> to a slice.	<code>s[:1] = []</code> <code>t[0:2] = []</code>	<code>s → [3]</code> <code>t → []</code>

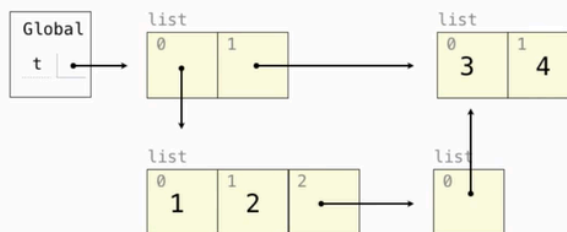
Lists in Lists in Lists in Environment Diagrams

```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```



[1, [...], 1, [...]]

```
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
```



[[1, 2, [[3, 4]]], [3, 4]]

