61A Lecture 14

Friday, October 3

Announcements

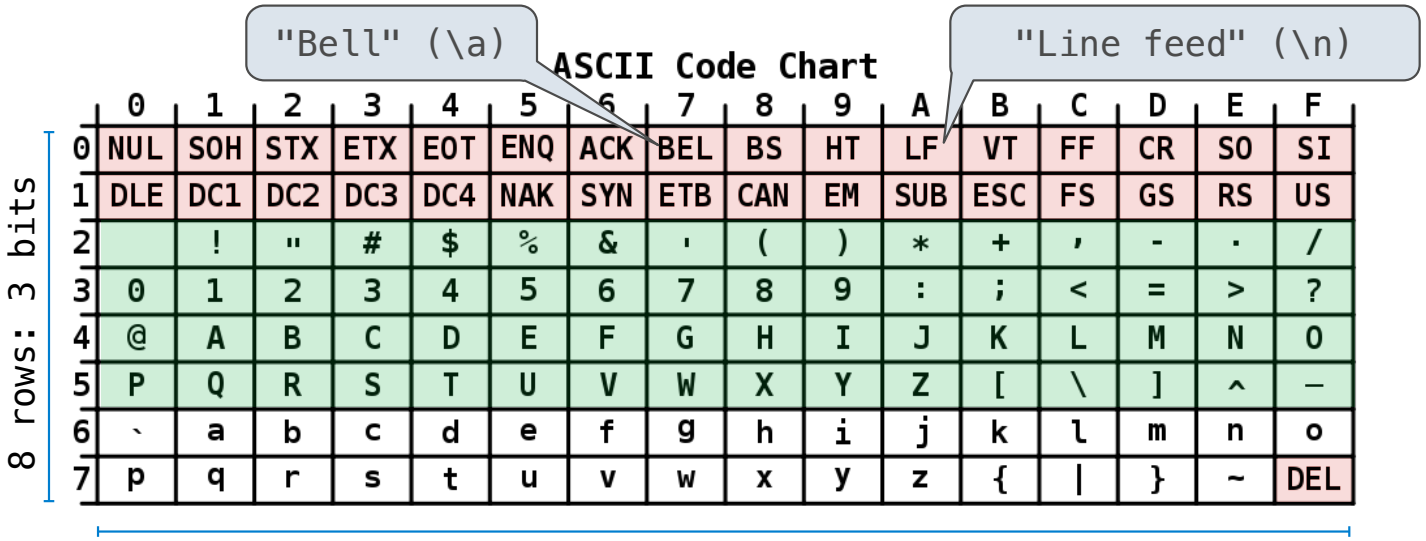
- Homework 4 due Tuesday 10/7 @ 11:59pm (It is small)
- Project 2 due Thursday 10/9 @ 11:59pm (It is BIG)
  - Project Party Monday 5pm–7pm in 271, 273, & 275 Soda
  - Extra credit point for submitting your project at least 24 hours before the deadline

Encoding Strings

(Bonus Material)

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

- 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

- 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

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)

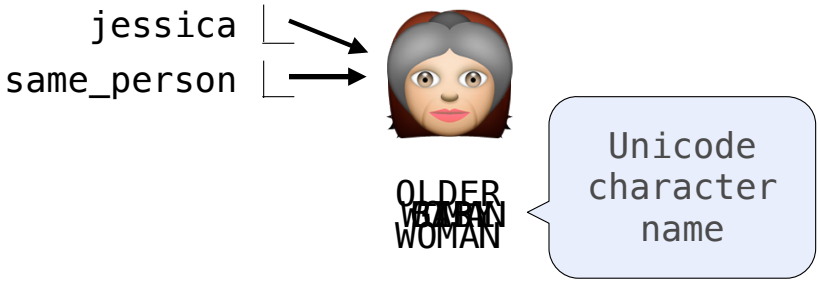
Mutation Operations

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}

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()
```

or

```
def mystery(s):
    s[2:] = []
```

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

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

Interactive Diagram

Tuples

(Demo)

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

- 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

`<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

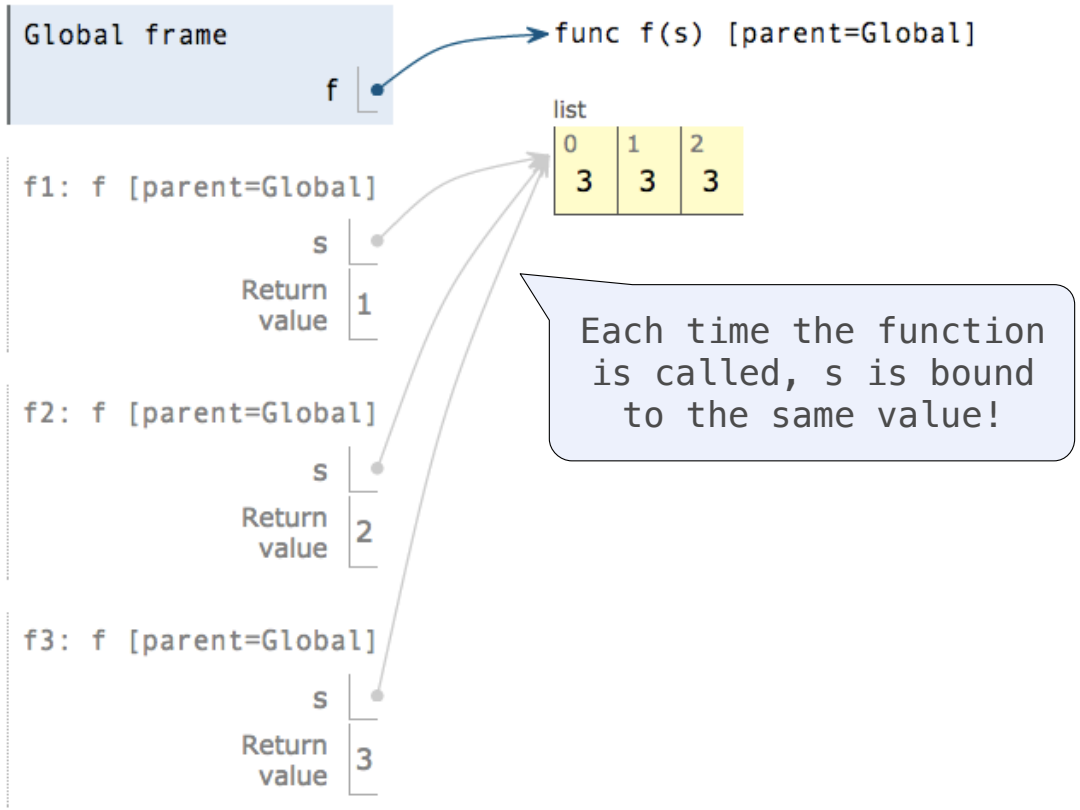
Identical objects are always equal values

(Demo)

Mutable Default Arguments are Dangerous

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