

61A Lecture 10

Monday, September 17

Sequence Iteration

```
def count(s, value):
    total = 0
    for elem in s:
```

Name bound in the first frame
of the current environment

```
        if elem == value:
            total = total + 1
    return total
```

For Statement Execution Procedure

```
for <name> in <expression>:
    <suite>
```

1. Evaluate the header <expression>, which must yield an iterable value.
2. For each element in that sequence, in order:
 - A. Bind <name> to that element in the first frame of the current environment.
 - B. Execute the <suite>.

Sequence Unpacking in For Statements

A sequence of
fixed-length sequences

```
>>> pairs = ((1, 2), (2, 2), (2, 3), (4, 4))
```

```
>>> same_count = 0
```

A name for each element in
a fixed-length sequence

Each name is bound to a value,
as in multiple assignment

```
>>> for x, y in pairs:
    if x == y:
        same_count = same_count + 1
```

```
>>> same_count
2
```

The Range Type

A range is a sequence of consecutive integers.*

..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ...

range(-2, 2)

Length: ending value - starting value

(Demo)

Element selection: starting value + index

```
>>> tuple(range(-2, 2))
(-2, -1, 0, 1)
```

Tuple constructor

```
>>> tuple(range(4))
(0, 1, 2, 3)
```

With a 0 starting value

* Ranges can actually represent more general integer sequences.

Membership & Slicing

The Python sequence abstraction has two more behaviors!

Membership.

```
>>> digits = (1, 8, 2, 8)
>>> 2 in digits
True
>>> 1828 not in digits
True
```

Slicing.

```
>>> digits[0:2]
(1, 8)
>>> digits[1:]
(8, 2, 8)
```

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"
```

Single- and double-quoted strings are equivalent

```
>>> '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.'
```

A backslash "escapes" the following character

"Line feed" character represents a new line

Strings are Sequences

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

An element of a string is itself a string!

Length. A sequence has a finite length.

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

```
>>> 'Berkeley' + ', CA'  
'Berkeley, CA'  
>>> 'Shabu ' * 2  
'Shabu Shabu '
```

String arithmetic is similar to tuple arithmetic

String Membership Differs from Other Sequence Types

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

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

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

The count method also matches substrings

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

the number of non-overlapping occurrences of a substring

Representing Strings: the ASCII Standard

Bonus Material

American Standard Code for Information Interchange

ASCII Code Chart															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS
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
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
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

Bonus Material

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

8B11	8B12	8B13	8B14	8B15	8B16	8B17	8B18
聲	聲	聳	聽	聒	聒	聒	聒
8B19	8B20	8B21	8B22	8B23	8B24	8B25	8B26
健	腰	腳	腴	腴	腴	腴	腸
8B27	8B28	8B29	8B30	8B31	8B32	8B33	8B34
眼	色	艷	艷	艷	艷	艷	艷
8B35	8B36	8B37	8B38	8B39	8B40	8B41	8B42
莖	莖	莖	莖	莖	莖	莖	莖
8B43	8B44	8B45	8B46	8B47	8B48	8B49	8B50
葱	葱	葱	葱	葱	葱	葱	葱

<http://ian-albert.com/unicode-chart/unicode-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 numbers and bytes

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

	00000000	0	
bytes	00000001	1	integers
	00000010	2	
	00000011	3	

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

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

Demo

13

Sequences as Conventional Interfaces

Consider two problems:

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

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

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

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

accumulate sum: ., ., ., ., 44.

14

Sequences as Conventional Interfaces

Consider two problems:

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

```

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

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

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

15

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

16

Accumulation and Iterable Values

Iterable objects give access to some elements in order.

Python-specific construct; less specific than a sequence

Many built-in functions take iterable objects as argument.

tuple	Return a tuple containing the elements
sum	Return the sum of the elements
min	Return the minimum of the elements
max	Return the maximum of the elements

For statements also operate on iterable values.

Demo

17

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>)`

Precise evaluation rule introduced in Chapter 4.

Demo

18

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

Like accumulate from Homework 2, but with iterable objects