

SICP-Spring2018

January 9, 2019

0.1 [U+5927] [U+4F6C] [U+9547] [U+697C]



###

[U+7B80] [U+4ECB] [U+4F2F] [U+514B] [U+5229] [U+5927] [U+5B66] CS61 [U+7CFB] [U+5217] [U+7B2C] [U+4E00] [U+FF0C] SICP [U+7684] Python [U+7248] [U+672C] [U+3002]

```
In [ ]: ### 1.3 control
- 1 python -i [U+4E86] [U+89E3] [U+4E00] [U+4E0B] interactive
- 2 pyton -m [U+4E86] [U+89E3] [U+4E00] [U+4E0B] -m -i
- 3 python -v
A statement is:
False in Python [U+FF1A] False 0 '' None
env is a sequence of frames
```

```
In [3]: from operator import mod,truediv,floordiv
mod(2013,5)
```

```
Out[3]: 3
```

```
In [5]: # Quotient
```

```
Out[5]: 1.0
```

0.1.1 1.4 Higher-order function

cs98 addtional Fibobacci seq [U+4E0E] [U+8D1D] [U+58F3] [U+FF1F] [U+6240] [U+6709] [U+9012] [U+5F52] [U+5747] [U+53EF] [U+4EE5] [U+7528] [U+975E] [U+9012] [U+5F52] [U+65B5] [U+81EA] [U+4E0A] [U+800C] [U+4E0B] [U+81EA] [U+4E0B] [U+800C] [U+4E0A] [U+7684] [U+5907] [U+5FD8] [U+5F55] function's domain [U+8FB9] [U+754C] [U+503C] function's range [U+9608] [U+503C] function's behavior one job DRY repeat generally **shared implention** [U+FF1A] **area solution** [U+51FD] [U+6570] [U+4F5C] [U+4E3A] [U+4ECE] [U+53C2] [U+6570] [U+4F20] [U+8F93] [U+8FDB] [U+53BB] [U+8FDD] [U+8FDB] [U+884C] generally $8/(4k-1)(4k-3) = \pi$ Function are first class:func can be manipulated as a value in program language Higher Order funtion:A function taht takes a function as an arg value or ret a func as a ret value

Lambda expression vs def: [U+690D] [U+7269] [U+5927] [U+6218] [U+50F5] [U+5C38] [U+62C9] [U+59C6] [U+8FBE]

In [9]: assert 3 < 2 , ' Falee'

```
-----
AssertionError                                                 Traceback (most recent call last)

<ipython-input-9-2baa5041699e> in <module>()
----> 1 assert 3 < 2 , ' Falee'

AssertionError:  Falee
```

In [11]: def make_adder(n):
 def adder(k):
 return k+n
 return adder

In [13]: make_adder(1)(10)

Out[13]: 11

In [15]: square = lambda x:x*x

0.1.2 1.5 Environment for H_O function

[U+7C7B] [U+4F3C] [U+4E8E] Python [U+7684] [U+88C5] [U+9970] [U+5668] [U+FF1F] [U+FF1F] [U+73AF] [U+5883] [U+7C7B] [U+4F3C] [U+4E8E] Linux fork() [U+51FD] [U+6570] frame [U+7684] [U+6982] [U+5FF5] [U+5F88] [U+91CD] [U+8981] [U+5173] [U+4E8E] [U+51FD] [U+6570] [U+7684] [U+5173]

0.1.3 1.6 Interation

fibnacci seq kth dif = kth+dif,kth

A call stamnets is: switch back env fx has a value only one ret [U+FF1A] fork()?? self references WAV Files

```
In [1]: def print_sum(x):
    print(x)
    def next_sum(y):
        return print_sum(x+y)
    return next_sum
print_sum(1)(3)(5)
```

```
1
4
9
```

```
Out[1]: <function __main__.print_sum.<locals>.next_sum(y)>
```

0.1.4 1.7 recursive function

call himself directly or not different frames keep diff argument in each call can be tricky:**Iteration is a special case of recursion!** Idea:figure out what state must be cantaintend by the iterative funtion converting iter to recursion: The state can be passed as arguments [U+9012] [U+5F52] [U+7ED3] [U+675F] [U+6761] [U+4EF6] [U+FF1A] [U+81EA] [U+8EAB] [U+6EE1] [U+8DB3] [U+6761] [U+4EF6] [U+7ED3] [U+675F] Mutual recursion Luhn sum recursion and iteration

```
In [2]: def split(n):
    return n//10,n%10
def sum_digits(n):
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return sum_digits(all_but_last) + last
def luhn_sum():
    pass
```

```
In [3]: sum_digits(2013)
```

```
Out[3]: 6
```

0.1.5 1.8 Tree recursion

```
./image/\begingroup \let \relax \relax \endgroup [Please insert \PrerenderUnicode{\unicode{36882}}]
```

[U+9012] [U+5F52] [U+7684] [U+91CD] [U+590D] [U+8BA1] [U+7B97] [U+592A] [U+591A] [U+5F88] [U+5230] 35 [U+5DF2] [U+7ECF] [U+6302] [U+4E86] by remembering results #### [U+89E3] [U+51B3] [U+597D] base case [U+95EE] [U+9898] [U+5176] [U+4ED6] [U+4EA4] [U+7ED9] [U+9012] [U+5F52] [U+FF08] [U+611F] [U+89C9] [U+5F88] [U+7C7B] [U+4F3]

```
In [13]: from ucb import trace
@trace
def cascade(n):
    if n < 10:
        pass
        #print(n) ## base case return None
    else:
        #print(n)
        cascade(n//10)
        #print(n)
def cascade_inverse():
    pass

@trace
def fib(n):
    if n<=1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

```
In [15]: #cascade(12345)
          #fib(10)
```

```
In [19]: ## countingh partitions
## description : [U+628A] [U+4E00] [U+4E2A] [U+6570] [U+5206]n[U+4E3A] [U+51E0] [U+4E2A] [U+6
def count_partition(n,m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0 :
        return 0
    else:
        with_m = count_partition(n-m,m)
        without_m = count_partition(n,m-1)
        return with_m + without_m
count_partition(6,4)
```

Out[19]: 9

0.1.6 1.9 Function Examples

Functional Abstractions



```
def square(x):           def sum_squares(x, y):  
    return mul(x, x)       return square(x) + square(y)
```

What does `sum_squares` need to know about `square`?

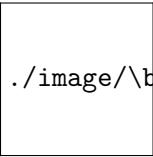
- | | |
|--|-----|
| • Square takes one argument. | Yes |
| • Square has the intrinsic name <code>square</code> . | No |
| • Square computes the square of a number. | Yes |
| • Square computes the square by calling <code>mul</code> . | No |

```
def square(x):           def square(x):  
    return pow(x, 2)       return mul(x, x-1) + x
```

If the name "square" were bound to a built-in function,
sum_squares would still work identically.



[U+51FD] [U+6570] [U+547D] [U+540D]



./image/\begingroup \let \relax \relax \endgroup [Please insert \PrerenderUnicode{\unichar{2146}}



./image/\begingroup \let \relax \relax \endgroup [Please insert \PrerenderUnicode{\unichar{2146}}

[U+53D8] [U+91CF] [U+547D] [U+540D]

- 1 repeated - 2 names can be long to help depument your code ##### Test Driven Development python -m doctest ex.py ##### function currying

Decorators

Enclosing scope

```
In [20]: def gcd(m,n):  
    '''Return the K largest  
    k , m, n are all pso  
    >>> gcd(12,8)  
    4  
    >>> gcd(16,12)  
    4  
    ...  
    if m == n:  
        return m  
    elif m < n:  
        return gcd(n,m)
```

```

    else:
        return gcd(m-n,n)

```

In [23]: gcd(4,3)

0.1.7 1.10 Data Abstraction

Data Abstraction

- Compound objects combine objects together
- A date: a year, a month, and a day
- A geographic position: latitude and longitude
- An *abstract data type* lets us manipulate compound objects
- Isolate two parts of any program that uses data:
 - How data are represented (as parts)
 - How data are manipulated (as units)
- Data abstraction: A methodology by which functions provide an *abstraction barrier* between *representation* and *use*

[U+6570] [U+636E] [U+62BD] [U+8C61] [U+7684] [U+610F] [U+4E49]
Pairs #### Abstration Barriers #### Dat a presentation

In [28]: `from operator import getitem`
`pair = [1,2]`
`x, y = pair`
`getitem(pair,0)`

Out[28]: 1

0.1.8 1.11 Containers

- List for in [U+7D22] [U+5F15] [U+7684] [U+5DE6] [U+95ED] [U+53F3] [U+5F00] [U+533A] [U+95F4] [-2,2) length: [U+53F3] [U+51CF] [U+5DE6] [U+4EE5] [U+504F] [U+79FB] [U+91CF] [U+4F5C] [U+4E3A] [U+89D2]
- List Comprehensions [U+63A8] [U+5BFC]
- string [U+7684] [U+5143] [U+7D20] [U+4ECD] [U+4E3A] string
- Dict limitation [U+FF1A] key [U+4E0D] [U+80FD] [U+4E3A] [U+53EF] [U+53D8] [U+7684] [U+7684] [U+6570] [U+4E3A] mutable

In [29]: `[2,7] + [1,2,3] * 2`

Out[29]: [2, 7, 1, 2, 3, 1, 2, 3]

In [36]: `pairs = [[1,2],[3,4],[5,6],[7,8]]`
`for x,y in pairs:`
 `print(x,y)`
`range(-2,2)`
`y = [1,2,3,4,5]`
`[x for x in y if x%2 == 0]`

```
1 2  
3 4  
5 6  
7 8
```

```
Out[36]: [2, 4]
```

```
In [37]: {x:x**x for x in range(10)}
```

```
Out[37]: {0: 1,  
          1: 1,  
          2: 4,  
          3: 27,  
          4: 256,  
          5: 3125,  
          6: 46656,  
          7: 823543,  
          8: 16777216,  
          9: 387420489}
```

0.1.9 1.12 mutable values

unicode standard

- 109000 charters
- 93 scripts enumeration of charter properties

```
mutation change dict.pop(key) ##### tuples are sequence but they are  
not mutable sequence tuple [U+4E0D] [U+80FD] [U+88AB] [U+6539] [U+53D8]  
[U+7528] [U+6237] [U+4FDD] [U+62A4] [U+6570] [U+636E] [U+7C7B] [U+4F3C] [U+4E8E] c  
const [U+FF1F] [U+FF1F] tuple [U+4E0D] [U+80FD] [U+6539] [U+53D8]  
[U+4F46] [U+662F] [U+5143] [U+7D20] [U+53EF] [U+4EE5] [U+6539] [U+53D8] [U+FF01] [U+FF01]
```

'is' and '=='

- is evaluate to True if both a and b evaluates the same object
- == evaluate to True if both a and b evaluates the same value identical object are always the same value ##### mutable values as default arg are dangerous

```
In [6]: from datetime import date  
        print('\a\a\a') #bell  
        a = 'A'  
        print(ord(a),hex(65))
```

```
65 0x41
```

```
In [10]: from unicodedata import lookup
        lookup('BABY')
        a = ([1,2,3],4,5)

        print(a)
```

```
([1, 2, 3], 4, 5)
```

```
In [14]: a[0][1] = 6
        print(a)
```

```
([1, 6, 3], 4, 5)
```

```
In [19]: def f(s = []):
            s.append(1)
            return len(s)
        for i in range(5):
            print(f()) really dangerous
```

```
1
2
3
4
5
```

0.1.10 1.14 mutable functions

nonlocal statement

0.1.11 1.15 oriented-object programming

Methods and Functions

Python distinguishes between:

- *Functions*, which we have been creating since the beginning of the course, and
- *Bound methods*, which couple together a function and the object on which that method will be invoked.

Object + Function = Bound Method

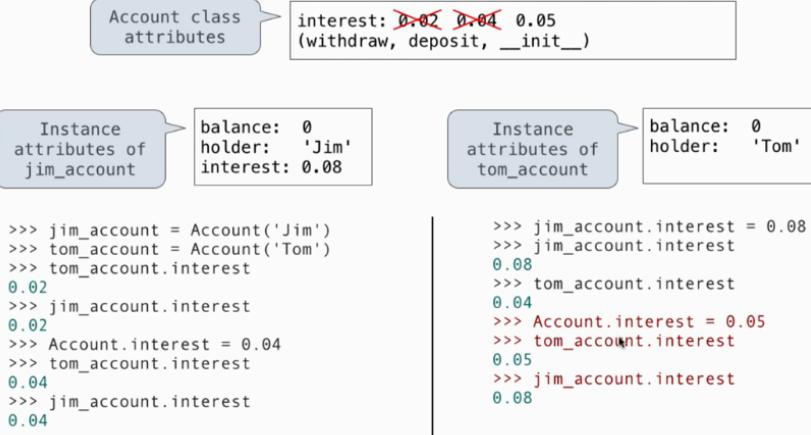
```
>>> type(Account.deposit)
<class 'function'>
>>> type(tom_account.deposit)
<class 'method'>

>>> Account.deposit(tom_account, 1001)
1011
```

invoking methods function and method

attribute [U+5F88] [U+5947] [U+602A] [U+4E3A] [U+4EC0] [U+4E48] [U+53EF] [U+4EE5] [U+4FDD] [U+5B58] [U+4E00]

Attribute Assignment Statements



21

0.1.12 1.6 inheritate

base class [U+7684] attribute [U+4E0D] [U+4F1A] [U+62F7] [U+8D1D] [U+5230] subclass - attribute

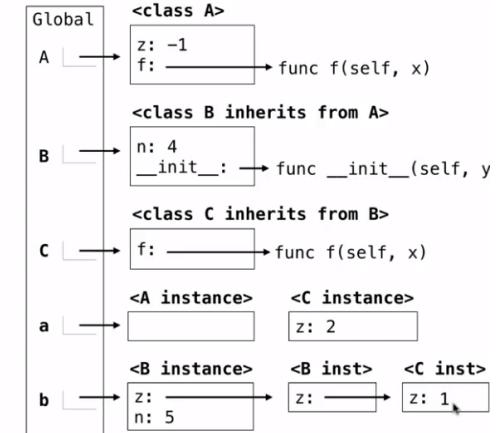
Inheritance and Attribute Lookup



```
class A:
    z = -1
    def f(self, x):
        return B(x-1)
    >>> C(2).n
    4

class B(A):
    n = 4
    def __init__(self, y):
        if y:
            self.z = self.f(y)
        else:
            self.z = C(y+1)
    True
    >>> a.z == C.z
    False

class C(B):
    def f(self, x):
        return x
    >>> Which evaluates
    >>> to an integer?
    b.z
    b.z.z
    b.z.z.z
    b.z.z.z.z
    None of these
    >>> a = A()
    >>> b = B(1)
    >>> b.n = 5
```



lookup
complicated inheritance

- complicated inheritance

0.1.13 1.7 Representation

`str` and `repr`

polymorphic functions [U+591A] [U+6001] [U+51FD] [U+6570]

- how implement this behavior ##### interface ##### special method names
- always with two underscores
- **add str booler float**
- [U+7C7B] [U+4F3C] [U+4E8E] [U+5B9E] [U+73B0] [U+591A] [U+6001] [U+6216] [U+8005] [U+91CD] [U+8F7D]

```
In [1]: >>> from fractions import Fraction
>>> half = Fraction(1,2)
>>> repr(half)
'Fraction(1, 2)'
>>> eval(repr(half))
Fraction(1, 2)
```

Out[1]: Fraction(1, 2)

0.1.14 1.18 Growth

measuring efficency

idea: remeber the result has been computed

- The consumption of Time
- Order of Growth
- constant does not affect
- The base of logarithm does not affetx \log_2 \log_{10} $O(b^n)$ $O(n^2)$ $O(n)$ $O(n^{0.5})$ $O(\log n)$ $O(1)$
- Exponentiation

b^n

```
In [21]: #### [U+597D] [U+597D] [U+7406] [U+89E3] [U+4E00] [U+4E0B] [U+4E0B] [U+9762] [U+4EE3] [U+7801]
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
def count(f):
    def counted(n):
        counted.call_count += 1
        return f(n)
    counted.call_count = 0
    return counted
def memo(f):
    cache = {}
    def memoized(n):
        if n not in cache:
            cache[n] = f(n)
        return cache[n]
    return memoized
print(fib(10))
fib = count(fib)
print(fib(10),fib.call_count)
counted_fib = fib
```

```

fib = memo(fib)
fib = count(fib)
print(fib(10),fib.call_count, counted_fib.call_count)

55
55 177
55 19 188

```

In [25]: `from ucb import trace`

```

@trace
def exp(b,n):
    if n == 0:
        return 1
    else:
        return b * exp(b,n-1)

@trace
def fast_exp(b,n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return square(fast_exp(b,n//2))
    else:
        return b * fast_exp(b,n-1)

```

In [26]: `exp(10,5)`

```

exp(10, 5):
    exp(10, 4):
        exp(10, 3):
            exp(10, 2):
                exp(10, 1):
                    exp(10, 0):
                        exp(10, 0) -> 1
                        exp(10, 1) -> 10
                        exp(10, 2) -> 100
                        exp(10, 3) -> 1000
                        exp(10, 4) -> 10000
    exp(10, 5) -> 100000

```

Out[26]: 100000

0.1.15 1.19 composition

- Linked list

- property methods
- Tree class
- Tree Mutation

In [28]: `assert 1 is 1`

0.1.16 1.20 set

`s.union s.intersection s.add()` ##### implementing sets ##### sets as linked liked list - searching an ordered list - set operations `def intersect(set1, set2)` - set mutation

0.1.17 1.21 Binary Tree

Binary search [U+FF1A] check the middle and eliminate half

Bnary search Trees:

- The biggest element in bst :[U+6700] [U+53F3]

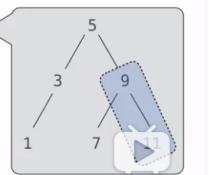
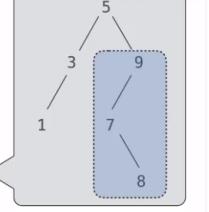
Discussion Questions

What's the largest element in a binary search tree?

```
def largest(t):
    if t.right is BTTree.empty_:
        return t.label
    else:
        return largest(t.right)
```

What's the second largest element in a binary search tree?

```
def second(t):
    if t.is_leaf():
        return None
    elif t.right is BTTree.empty_:
        return largest(t.left)
    elif t.right.is_leaf():
        return t.label
    else:
        return second(t.right)
```



- The second biggest Tree ##### membership in BST [U+6BCF] [U+6B21] [U+6298] [U+534A] [U+65F6] [U+95F4] [U+590D] [U+6742] [U+5EA6] [U+4E3A] logn

In [29]: `from random import shuffle`

0.1.18 1.22 Data Examples

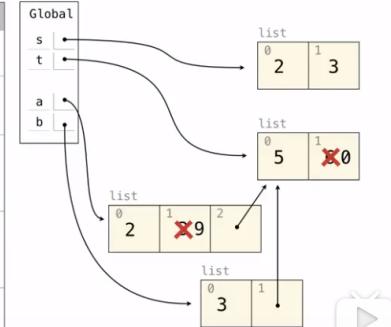
- List
- append extend addation %slicing

Lists in Environment Diagrams



Assume that before each example below we execute:
`s = [2, 3]
t = [5, 6]`

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



####

- list in list environment diagram

- frame
- parent frame #### objects

Land Owners



Instance attributes are found before class attributes; class attributes are inherited

```

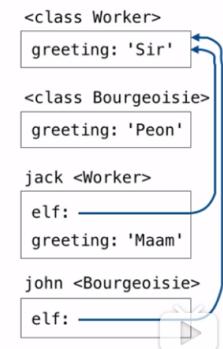
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
    
```

```

>>> Worker().work()
'Sir, I work'
>>> jack
Peon
>>> jack.work()
'Maam, I work'
>>> john.work()
Peon, I work
'I gather wealth'
>>> john.elf.work(john)
'Peon, I work'
    
```



re-

- instance cursive link

morse code

```
In [36]: t = [1,2,3]
         t[1:3] = [t]
         t.extend(t)
         print(t,len(t[1]))
```

[1, [...], 1, [...]] 4

```
In [ ]: def Worker:
         g
```

0.1.19 1.24

scheme

0.1.20 1.25 Exceptions

Exceptions



A built-in mechanism in a programming language to declare and respond to exceptional conditions

Python raises an exception whenever an error occurs.

Exceptions can be handled by the program, preventing the interpreter from halting.

Unhandled exceptions will cause Python to halt execution and print a stack trace.

Mastering exceptions:

Exceptions are objects! They have classes with constructors.

They enable non-local continuations of control:

If f calls g and g calls h, exceptions can shift control from h to f without waiting for g to return.

exceptions

####

rasing exception - assert , assert python -0 ex.py

TypeErr KeyErr NameErr RuntimeErr([U+9012] [U+5F52] [U+8FC7] [U+6DF1] [U+7B49])

Try statement

WWPD

Reduce[U+FF1A]

- reduce a sequence to a value #### Sierpinski's triangle
-

```
In [38]: raise TypeError("Bad")
      def f():
          f()
```

TypeError

Traceback (most recent call last)

```
<ipython-input-38-3b635fc5af21> in <module>()
----> 1 raise TypeError("Bad")
      2 def f():
```

```
3      f()
```

```
TypeError: Bad
```

```
In [41]: from operator import add  
reduce(add,[1,2,3],4)
```

```
-----  
NameError                                     Traceback (most recent call last)
```

```
<ipython-input-41-6e8e7f9b87a9> in <module>()  
      1 from operator import add  
----> 2 reduce(add,[1,2,3],4)
```

```
NameError: name 'reduce' is not defined
```

0.1.21 1.26 calculator

Interpreters

parsing

0.1.22 1.27 Interpreters

Base cases: - scheme evaluation - logical special forms - Quotation

0.1.23 1.28 Tail calls

Dynamic scope

lexical scope

tail recursion

functional programming

[U+5C3E] [U+8C03] [U+7528] linear recursive can always be re-write as tail call

0.1.24 Map and Reduce Google [U+8BBA] [U+6587] [U+5927] [U+4F5C]

An analogy: programs specific machine

```
In [45]: list(map(abs,[1,2,3,-4]))  
       reduce()
```

```
Out[45]: [1, 2, 3, 4]
```

0.1.25 1.29 Macros

0.1.26 1.30 Iterators

Data processing

- process sequential data
- sequence interface
- has a finite known length
- allows element selection
- big data processing
- Distributed and parallel computing ##### Iterators
- iter return an iterator
- next return the next element in an iterator ##### For statements
- when exe for statements ,iter returns an iterator and next provides each item
- A StopIteration exception is raised when next is called on empty iterator ##### bulit-in iterator fnctions [U+6CE8] [U+610F] [U+554A] iterable

Built-in Functions for Iteration

Many built-in Python sequence operations return iterators that compute results lazily.

<code>map(func, iterable):</code>	Iterate over func(x) for x in iterable
<code>filter(func, iterable):</code>	Iterate over x in iterable if func(x)
<code>zip(first_iter, second_iter):</code>	Iterate over co-indexed (x, y) pairs
<code>reversed(sequence):</code>	Iterate over x in a sequence in reverse

To view the contents of an iterator, place the resulting elements into a container

<code>list(iterator):</code>	Create a list containing all x in iterator
<code>tuple(iterator):</code>	Create a tuple containing all x in iterator
<code>sorted(iterator):</code>	Create a sorted list containing x in iterator

- map() filter(f,iterable) zip() reversed()

The screenshot shows a terminal window on the left and a video player window on the right. The terminal window displays Python code being run:

```

~/lec$ python3 -i ex.py
>>> m = map(double, range(3, 7))
>>> f = lambda y: y >= 10
>>> t = filter(f, m)
>>> next(t)
** 3 => 6 **
** 4 => 8 **
** 5 => 10 **
10
>>> next(t)
** 6 => 12 **
12
>>> list(t)
[]
>>> list(filter(f, map(double, range(3, 7))))
** 3 => 6 **
** 4 => 8 **
** 5 => 10 **
** 6 => 12 **
[10, 12]
>>>

```

The video player window shows a video frame of a person speaking, with a play button icon and the text "ex.py" 3L, 66C written at the bottom.

Generator and generator func

- yield ##### generator and iterator
- genor can Yield from iterator
- yield from [U+8BED] [U+6CD5]

In [46]: s = [1, 2, 3]
t = iter(s)

In [49]: next(t)
list(t) ## [U+7ED3] [U+675F] [U+8FED] [U+4EE3]

StopIteration Traceback (most recent call last)

```

<ipython-input-49-1376cbf80061> in <module>()
----> 1 next(t)
      2 list(t)

```

StopIteration:

In [51]: d = {'1': 2, '2': 'two'}
k = iter(d)

In [52]: next(k)

Out[52]: '1'

In [53]: d.pop('2')

```

Out[53]: 'two'

In [66]: ### attention
t = [1,2,3,2,1]
print(t == reversed(t),t == list(reversed(t)))

d = {'1':'one','2':'two'}
items = zip(d.keys(),d.values())
print(list(items))

## generator
def evens(start,end):
    even = start + (start % 2)
    while even < end:
        yield even
        even += 2
g = evens(2,10)
print(list(g))

def cntdown(k):
    if k > 0:
        yield k
        yield from cntdown(k-1)
for k in cntdown(3):
    print(k)
print(list(cntdown(10)))

False True
[('1', 'one'), ('2', 'two')]
[2, 4, 6, 8]
3
2
1
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

```

0.1.27 1.31 streams

efficient sequence operations

all() [U+51FD] [U+6570] [U+600E] [U+4E48] [U+64CD] [U+4F5C] [U+7684]

```
In [4]: all(map(lambda y: 1 % y,range(1,6)))
```

```
Out[4]: False
```

Apache Spark



Apache Spark is a data processing system that provides a simple interface for large data

- A Resilient Distributed Dataset (RDD) is a collection of values or key-value pairs
- Supports common UNIX operations: `sort`, `distinct` (`uniq` in UNIX), `count`, `pipe`
- Supports common sequence operations: `map`, `filter`, `reduce`
- Supports common database operations: `join`, `union`, `intersection`

All of these operations can be performed on RDDs that are partitioned across machines

Romeo & Juliet

Two households, both alike in dignity,
In fair Verona, where we lay our scene,
From ancient grudge break to new mutiny,
Where civil blood makes civil hands more亏亲
From forth the fatal loins of these two foes
A pair of star-cross'd lovers take their life;
Whole misadventur'd piteous overthrows
Do with their death bury their parents' strife.
The fearful passage of their death-mark'd love,
And the continuance of their parents' rage,
Whereon all depends. O, I could remove
Is now the two hours' traffic of our stage:
The which if you with patient ears attend,
What here shall miss, our toil shall strive to mend.

0.1.28 1.32 declarative programming

DataBase Management system DBMS declarative language

Declarative Programming



In **declarative languages** such as SQL & Prolog:

- A "program" is a description of the desired result
- The interpreter figures out how to generate the result

In **imperative languages** such as Python & Scheme:

- A "program" is a description of computational processes
- The interpreter carries out execution/evaluation rules

```
create table cities as
  select 38 as latitude, 122 as longitude, "Berkeley" as name union
  select 42,          71,           "Cambridge"      union
  select 45,          93,           "Minneapolis";
```

Cities:

Latitude	Longitude	Name
38	122	Berkeley
42	71	Cambridg
45	93	Minneapol

Region	Name
west coast	Berkeley
other	Minneapol
other	Cambridg

```
select "west coast" as region, name from cities where longitude >= 115 union
select "other".           name from cities where lonatitude < 115;
```



arithmetic

1.33 Tables pass ### 1.34 aggregation - SQL [U+4E13] [U+5C5E] - pass #### 1.35 DataBase - pass
1.36 Distributed Data - Unix - Computer System - Operate System - Networks - DataBase - Dis sys - Hide complexity but retain flexablity - stdin ----> process ----> stdout | std err

- sys.stdin sys.stdout
- Big Data
- Apache Spark

- MapReduce
- An important early distributed prossing system developed by google



MapReduce Applications

An important early distributed processing system was MapReduce, developed at Google

Generic application structure that happened to capture many common data processing tasks

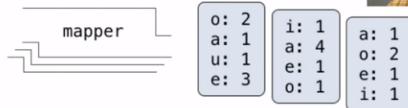
- Step 1: Each element in an input collection produces zero or more key-value pairs (map)
- Step 2: All key-value pairs that share a key are aggregated together (shuffle)
- Step 3: The values for a key are processed as a sequence (reduce)

Early applications: indexing web pages, training language models, & computing PageRank



MapReduce Evaluation Model

Google MapReduce
Is a Big Data framework
For batch processing



Reduce phase: For each intermediate key, apply a *reducer* function to accumulate all values associated with that key

- All key-value pairs with the same key are processed together
- The reducer yields zero or more values, each associated with that intermediate key



0.1.29 1.37 Natural language

Ambiguity

- Programs must be written for people to read! ---> preface of SICP
- Syntax Trees
- buffalo
- Recursive Tree
- Grammars
- Parsing
- Learning

end

currying rational odd compound civil Comprehensions closure hierarchical polymorphic arbitrary scheme dialate quotient liberally useless spatial lexical analysis tokens tail middlemen macros evaluation Vote cast declarative imperative arithmetic ambiguity intimidate invoke

[U+67EF] [U+91CC] [U+5408] [U+7406] [U+7684] [U+5947] [U+590D] [U+5408]
[U+56FD] [U+5185] [U+63A8] [U+5BFC] [U+5173] [U+95ED] hierarachical [U+591A] [U+6001]

[U+968F] [U+610F] [U+65B9] [U+6848] dialate [U+5546] [U+5BBD] [U+677E] [U+65E0] [U+7528]
[U+7A7A] [U+95F4] [U+7684] lexical [U+5206] [U+6790] [U+4EE4] [U+724C] [U+5C3E] [U+5DF4]
[U+4E2D] [U+95F4] [U+5546] [U+5B8F] [U+8BC4] [U+6D4B] [U+6295] [U+7968] [U+9648] [U+8FF0]
[U+52BF] [U+5728] [U+5FC5] [U+884C] [U+7B97] [U+672F] [U+6B67] [U+4E49] [U+5A01] [U+5413]
[U+8C03] [U+7528]