Table of Contents

Virtual Environment

```
Python
#global
pip install virtualenv
pip list
which python
mkdir []
cd
virtualenv project1_env
virtualenv -p usr/bin/python2.6 py26_env #define python versino
source project1_env/bin/activate #activate
pip install ...
pip freeze --local > requirements.txt #save local environment packages list
pip install -r requirements.txt #install according to the txt file
deactivate #back to global
rm -rf #get rid of it
```

useful commands

```
Python
python3 -i lab00.py          //open interactive shell with this module
python3 -m doctest lab00.py //run doctests inside the file

//doctest example
"""
>>> twenty_nineteen()
2019
"""

```

Expressions

```
Python
shakes = open('shakespeare.txt')    #open file
text = shakes.read().split()
text.count('the')                   #count the number of apperence 'the'
words = set(text)
'the' in words                      #the value is a boolean 'True'

'draw'[::-1]                        #reverse the word, the last :-1 means step -1 (when it's neg
                                    #It means starting from the end
w = "the"
words = set(open('/usr/share/dict/words').read().split())   #open the default dictionary in mac
{w for w in words if w[::-1] == w and len(w) == 4}          #evaluated to a list of palindrome

7//4    #divide 7 by 4 and floor the result
```

[return to the top](#)

Functions

```python
ctrl+l //clear the screen

from math import pi
from math import sin
sin(pi/2)

f = max
f(1,2,3)    //return 3

from operator import add, mul
```

Ways to bind a name

- import
- assignment
- def statement

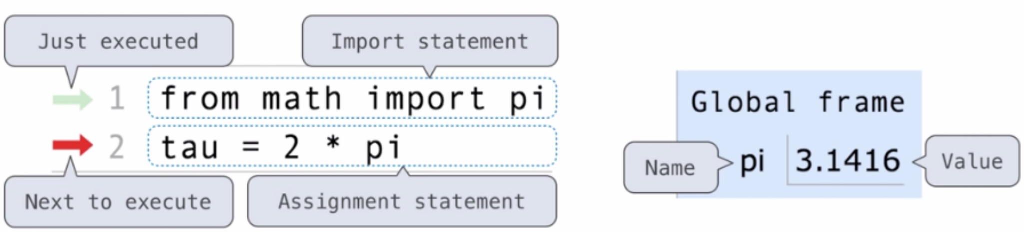
"def" statement e.g. `python def square(x): return mul(x, x)`

Types of Expressions

- Primitve expressions(2 add 'hello')
    - Number or Numeral (e.g. 2)
    - Name (e.g. add)
    - String (e.g. 'hello')

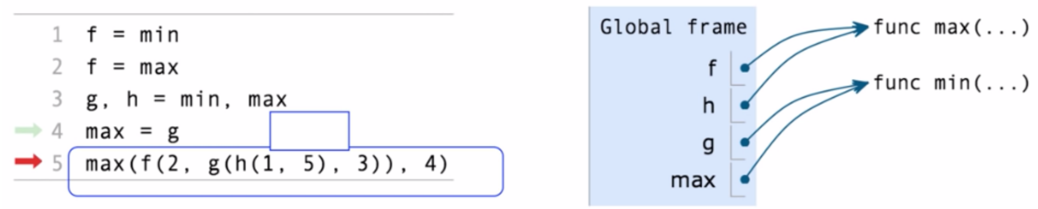
- Call expressions(max(2, 3))
    - operator (e.g. max)
    - operand(e.g. 2 | 3)

Environment Diagrams



Execution rule for assignment statements:

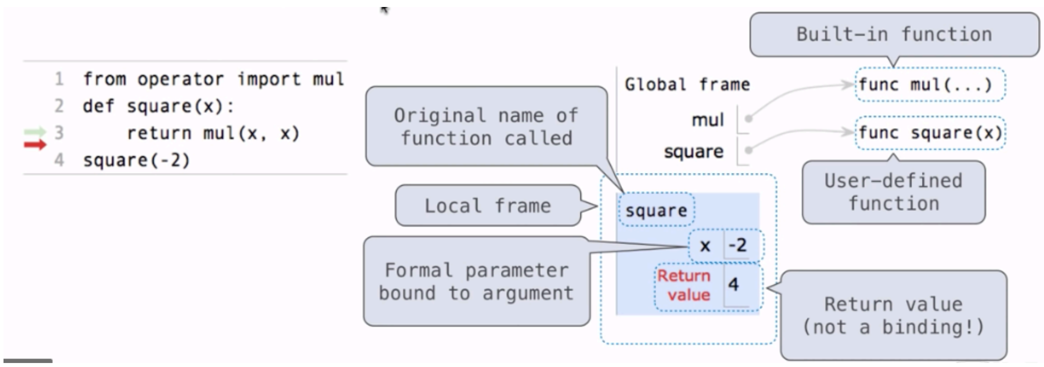
1. Evaluate all expressions to the right of = from left to right.
2. Bind all names to the left of = to the resulting values in the current frame.

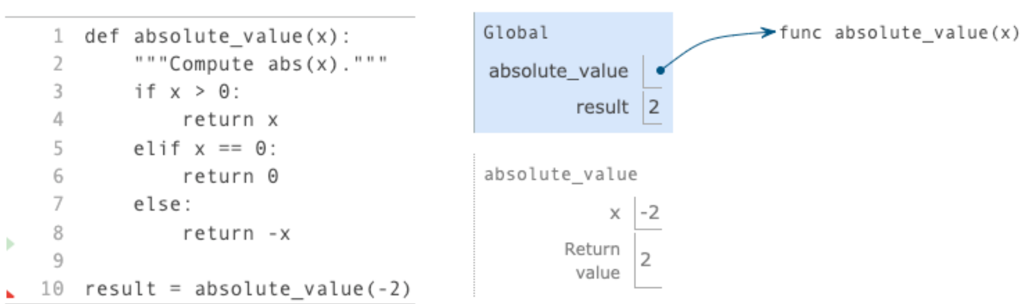


Functions

Means of Abstraction

- Assignment
- Function definition





[return to the top]

Testing

Assertions

Python

```python
assert fib(8) == 13, 'The 8th Fib number should be 13'
```

If the expression is - True, nothing will happen - False, it will cause an error, halt the execution and print the message

Doctests

```python
"""
>>> sum_nat(10)
55
>>> sum_nat(100)
5050
"""
```

Run Tests

1. Run all the tests

```python
>>> from doctest import testmod
>>> testmod() //run all the tests
```

1. run specific function test

```python
>>> from doctest import run_docstring_examples
>>> run_docstring_examples(sum_nat, globals(), True)    //sum_nat: function name, globals(): get
```

1. Run all the tests in a file

```
python3 -m doctest <python_source_file>
```

[return to the top](#)

Control

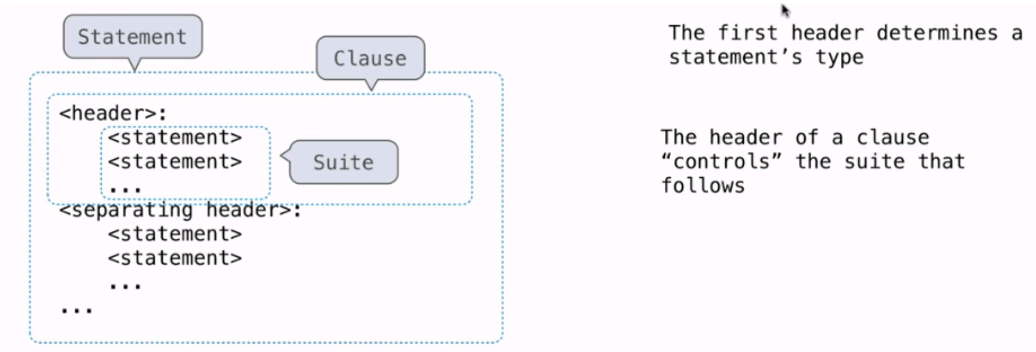
Print

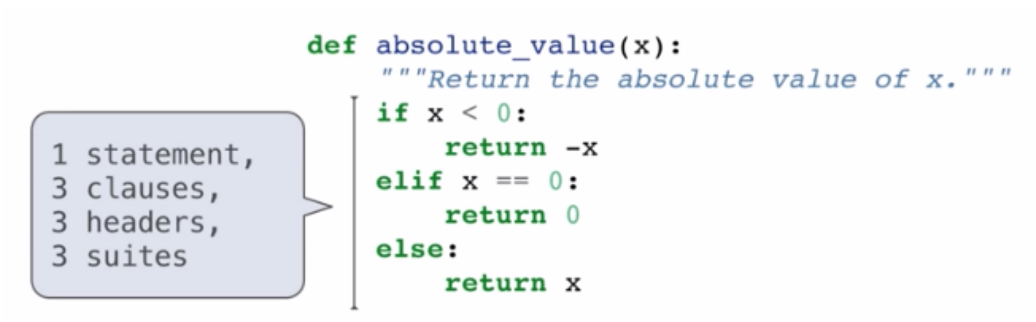
```python
print(1,2,3)
1 2 3

print(None, None)
None None

print(print(1), print(2))
1
2
None None
```

Miscellaneous Python Feature

```python
2013 // 10  //truediv div and ignore the reminder (floordiv)
2013 % 10   //mod(2013, 10)
```

Compound Statements





[return to the top]()

Lambda

Lambda Expressions

```python
(lamdda x: x*x)(3)      //call lambda function with arguement 3

square = lambda x: x*x
square(4)               //return 16
```

Difference between def ~ and lambda - function has a name when using def

```python
//using lambda
>>> square
<function <lambda> at 0x1003c1bf8>

//using def
>>> square
<function square at 0x10293e730>
```

return to the top

week1 miscellaneous

```python
>>>19 and 21
21
```

- inner call goes first because operands must be evaluated before calling a function

```python
def yes(guess):
    if guess == 'yes':
        return 'yes'
    return 'no'

def go(x):
    return x + yes(x)

go(go('yes'))

"""
f global
f1 go(inside)
f2 yes
f3 go(outside)
f4 yes
"""
```

- There is no quotes for the output of print function but there is quotes if a string is returned

```python
>>> print('fuck')
fuck

>>> 'fuck'
'fuck'
```

- There is no difference between single and double quotes in python

''(empty string), 0, False, None means False

- always prefer to show the latest one

- show nothing if None
- show only what evaluated
- comparison operator(like >) has higher priorities than the keywords: and, or

Python

```python
>>> -3 and True
True
>>> True and -3
-3

>>> False or None

>>> None or False
False

>>> True or 3
True

>>> 3 or True
3

>>> False and ''
False

>>> '' and False
''

>>> 1 or 0==0
1
```

- variables passed in as parameters can be changed directly

Python

```python
def say(s0):
    for i in range(10):
        s0 += 1
    return s0
```

- print

Python

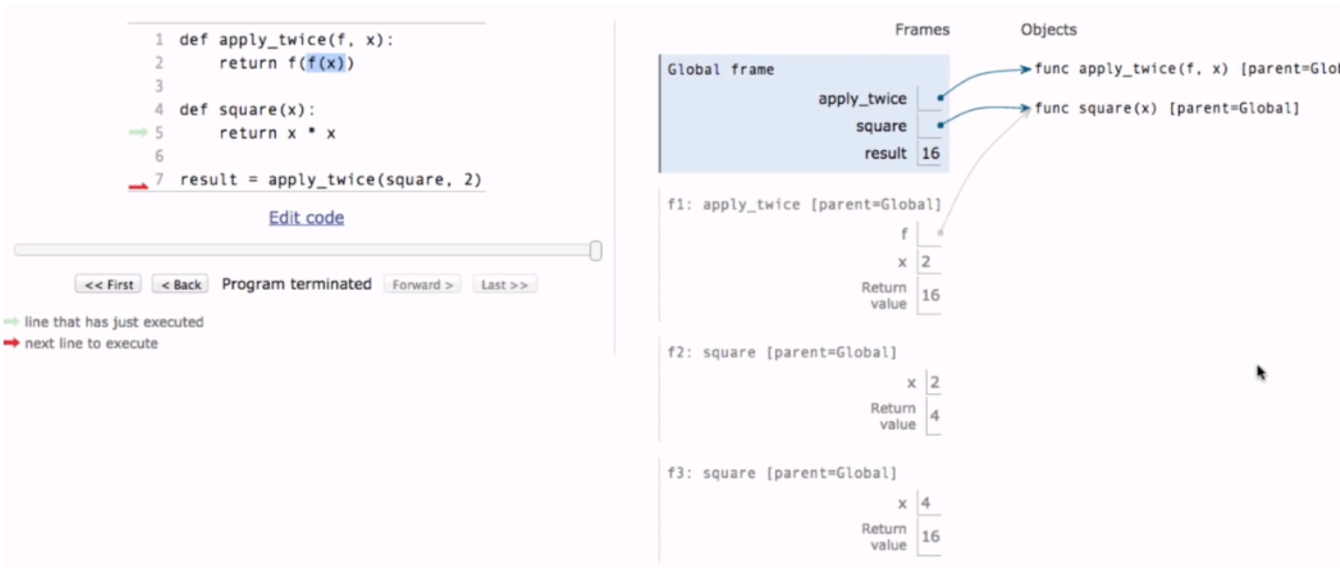
```python
>>> print(10, 20)
10 20
```

return to the top

Higher-order function:

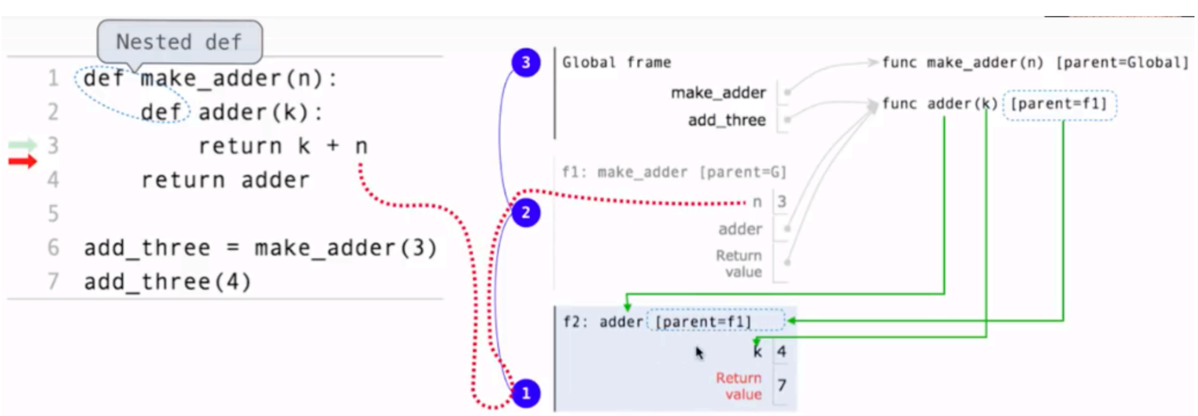
- A function that takes a function as an arguement or returns a function

- A function's domain is the set of all inputs it might possibly take as arguments.
- A function's range is the set of output values it might possibly return.
- A function's behavior is the relationship it creates between input and output.



Python

```python
#return a function
"""
>>> adder = make_adder(3)
>>> adder(4)
7
"""

def make_adder(n):
    def adder(k):
        return k + n
    return adder



>>> make_adder(1)(3)
>>> 4
```



[return to the top](#)

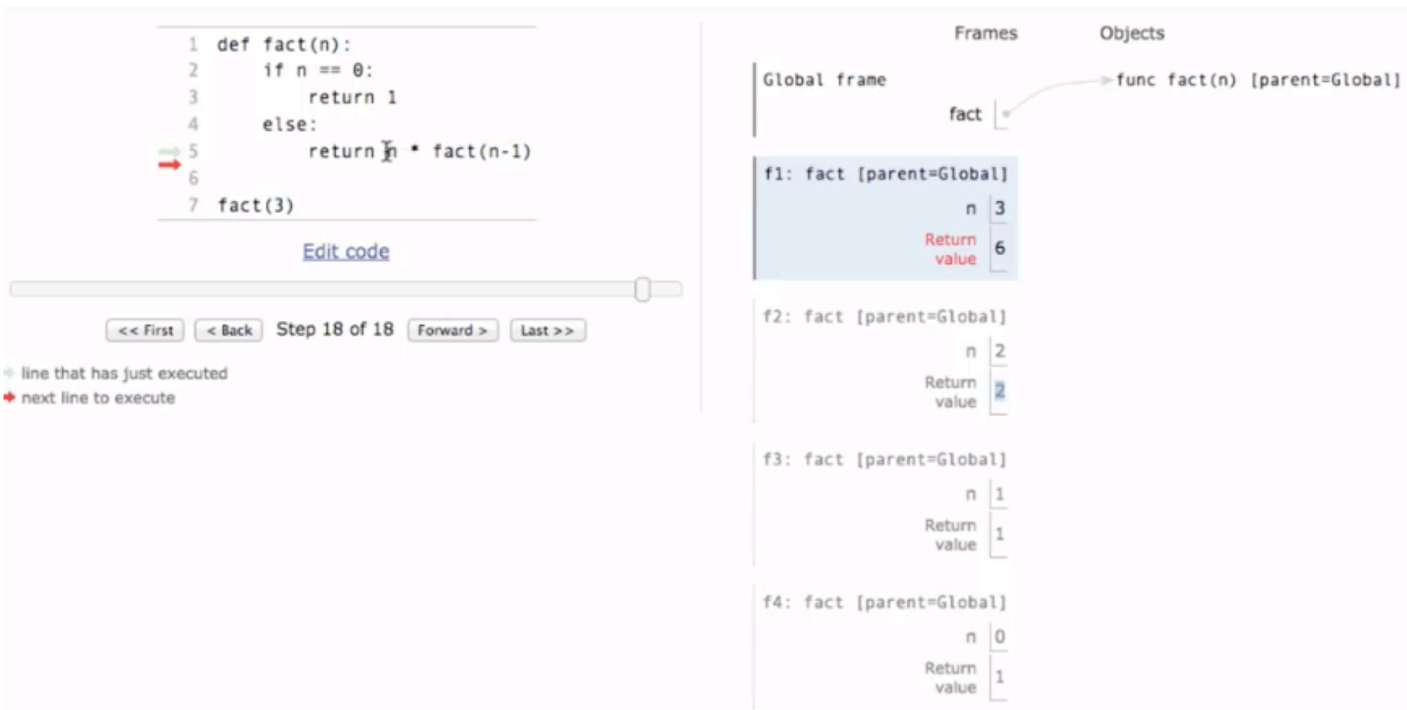
Recursive Functions

- A function is called recursive if the body of that function calls itself, eighter directly or indirectly.

Python
```python
def fact_iter(n):
    total, k = 1, 1
    while k <= n:
        total, k = total*k, k+1
    return total
```

‘$n! = \prod_{k=1}^{n} k$‘

Python
```python
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```



Tree Recursion

```python
#like a true
def fib(n):
    if n == 1:
        return 1
    elif n == 0:
        return 0
    else:
        return fib(n) + fib(n-1)
```

trace decorator

```python
from ucb import trace
@trace #???
def fib(n):
    if n == 1:
        return 1
    elif n == 0:
        return 0
    else:
        return fib(n) + fib(n-1)

"""trace enable tracing for every step like:
>>> fib(0)
    fib(0):
    fib(0) -> 0
    1
"""
```

Mutual Recursion

The Luhn Algorithm

- Used in the numbers of credit cards
- If any digit is changed, the sum won't be a multiple of 10

original	1	3	8	7	4	3	sum
result	2	3	1+6=7	7	8	3	30

- Start from the right most digit
- Double the value of every second digit
- If product is greater than 9 then sum the 2 digits
- Take the sum of all the digits, it is a multiple of 10

```python
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(n):
    if n < 10:
        return n
    else:
        all_but_last, last = split(n)
        return last + luhn_sum_double(all_but_last)

def luhn_sum_double(n):
    all_but_last, last = split(n)
    luhn_digit = sum_digits(2 * last)
    if n < 10:
        return luhn_digit
    else:
        return luhn_sum(all_but_last) +luhn_digit
```

- Be able to convert between iteration and recursion

Cascade

```python
def cascade(n):
    if n < 10:
        print(n)
    else:
        print(n)
        cascade(n//10)
        print(n)
"""
>>> cascade(5)
12345
1234
123
12
1
12
123
1234
12345
"""
```

```python
def invese_cascade(n):
    grow(n)
    print(n)
    shrink(n)

def f_then_g(f, g, n):
    if n:
        f(n)
        g(n)

grow = lambda n: f_then_g(grow, print, n // 10)
shrink = lambda n: f_then_g(print, shrink, n // 10)
"""
>>> inverse_cascade(4)
1
12
123
1234
123
12
1
"""
```

Count Partitions

2 + 4 = 6

1 + 1 + 4 = 6

3 + 3 = 6

1 + 2 + 3 = 6

1 + 1 + 1 + 3 = 6

2 + 2 + 2 = 6

1 + 1 + 2 + 2 = 6

1 + 1 + 1 + 1 + 2 = 6

1 + 1 + 1 + 1 + 1 + 1 = 6

Divide and Conquer

- include 4
- not include 4

```python
"""
>>> count_partitions(6, 4)
9
"""

def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:    #If m reaches 0 first, it will rely on the first term to add all 1s into the
        return 0

    return count_partitions(n - m, m) + cout_partitions(n, m - 1)
```

[return to the top](#)

Data Abstraction

- Compound objects combine objects together
    - A data: a year, a month, and a day
- An abstract data type lets us manipulate compound objects as units
- 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 enforce an abstraction barrier between representation and use
- Terminology
    - ADT : Abstract Data Type ```python import fractions import gcd #constructor def rational(n, d): g = gcd(n, d) return [n//g, d//g]

alternative way, instead of list

def rational(n, d): g = gcd(n, d) def select(name): if name == 'n': return n//g: elif name == 'd': return d//g return select

selector

def numer(x): return x[0]

selector:

def denom(x): return x[1]

def mul_rational(x, y): return rational(numer(x) * numer(y), denom(x) * denom(y))

def equal_rational(x, y): return numer(x) * denom(y) == numer(y) * denom(x)

```

#### Pairs
```python
>>> pair = [1, 2]

>>> x, y = pair #unpacking a list

>>> from operator import getitem
>>> getitem(pair 0)
1
>>> getitem(pair 1)
2
```

Abstraction Barriers

Parts of the program that...	Treat rationals as...	Using...
Use rational numbers to perform computation	whole data values	add_rational, mul_rational rationals_are_equal, print_rational
Create rationals or implement rational operations	numerators and denominators	rational, numer, denom
Implement selectors and constructor for rationals	two-element lists	list literals and element selection

Implementation of lists

[return to the top](#)

Built-in Types

Lists

Python

```python
digits = [1, 2]
from operator import mul,add
>>> add[2, 7] + mul(digits, 2)
[2, 7, 1, 2, 1, 2]
>>> [2, 7] + digits * 2
[2, 7, 1, 2, 1, 2]
```
```

Containers

```python
>>> 1 in digits
True

>>> 1 not in digits
False

>>> [1, 7] in digits
False

>>> [1, 2] in [3, [1, 2], 4]
True
>>> [1, 2] in [3, [[1, 2]], 4]
False
```

For Statements

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

#unpacking in for
for x, y in pairs:
    if x == y:
        print(1)
```

Range

- a sequence of consecutive integers ```python >>> List(range(-2, 2)) [-2, -1, 0, 1]

for _ in range(3): #Don't care about the number print('Go Bears!') ```

List Comprehensions

```python
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'm', 'n', 'o', 'p']
>>> [letters[i] for i in [3, 4, 6, 8]]
['d', 'e', 'm', 'o']

>>> odds = [1, 3, 5, 7, 9]
>>> [x+1 for x in odds if 25 % x == 0]
[2, 6]
```

Strings

```python
>>> exec('curry = lambda f: lambda x: lambda y: f(x, y)')
>>> curry
'curry = lambda f: lambda x: lambda y: f(x, y)'

>>> """The highness"""
'The highness'

>>> city = 'Berkeley'
>>> city[3]
'k' #no character, only string

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

Dictionaries

- No order at all ```python >>> n = {'a':1, 'b':2} >>> n.keys() >>> n.values() >>> n.items()

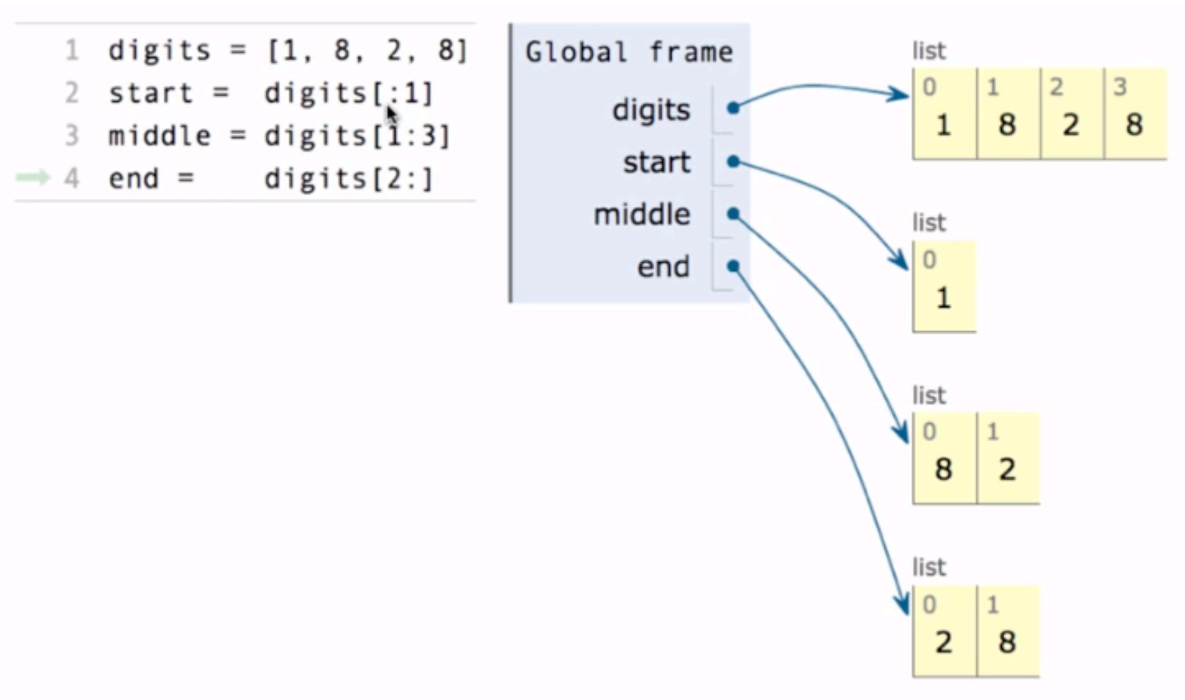
> > > items*list = [('a', 1), ('b', 2), ('c', 3)] a = dict(items*list) a['a'] 1
> > >
> > > a.get['a', 0] #default 0
> > >
> > > {x:x*x for x in range(10)} {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
> > >
> > > {[1]: 2} #error ``` [return to the top]

Trees

Slicing

```python
>>> odds = [3, 5, 7, 9, 11]
>>> odds[:3] = [3, 5, 7]
```

slicing creates new values



Processing Container Values

```python
>>> sum([2,3,4], 1)     #Doesn't work with string, 1 is starting value not index
7
>>> sum([2,3], [4]], [])
[2, 3, 4]

>>> max(0, 1, 2, 3, 4)
4
>>> max(range(5))
4
>>> max(range(10), key=lammbda x: 7-(x-4)*(x-2))
3
>>> bool(5)
True

>>> all(range(0, 5)}    #return True is every element bool(e) is True
>>r all(range(1, 5)}
True
```

Tree

```python
def tree(label, branches=[]):
    for branch in branches:
        assert is_tree(branch), 'branches must be trees'
    return [label] + list(branches)

def is_tree(tree):
```

```
    if len(tree) < 1 or type(branch) != list:
        return False
    for branch in branches(tree):
        if not is_tree(branch):
            return False

    return True

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:]

def is_leaf(tree):
    return not branches(tree)

def fib_tree(n):
    if n <= 1:
        return tree(n)
    else:
        left, right = fib_tree(n-2), fib_tree(n-1)
        return tree(label(left)+label(right), [left, right])

def count_leaves(t):
    if is_leaf(t):
        return 1
    else:
        return count([count_leaves for b in branches(t)])

def leaves(tree):
    if is_leaf(tree):
        return [label(tree)]
    else:
        sum([leaves(l) for l in branches(tree)], [])

#only increment leaves
def increment_leaves(t):
    if is_leaf(t):
        return tree(label(t)+1)
    else:
        bs = [increment_leaves(b) for b in branches(t)]
        return tree(label(t), bs)

def increment_leaves(t):
    return tree(label(t)+1, [increment(b) for b in branches(t)])

def print_tree(t, indent=0):
    print('  ' * indent + str(label(t)))
    for b in branches(t):
        print_tree(b, indent+1)

```

```
>>> print(' ' * 5 + str(5))
     5
```

[return to the top](#)

Mutable Sequence

Objects

```python
>>> from datetime import date
>>> today = date(2015, 2, 20)
>>> final = date(2015, 5, 12)
>>> str(freedom - today)
'81 days, 0:00:00'
>>> today.year
2015
>>> today.month
2
>> today.strftime('%A %B %d')
'Friday February 20'
```

- A type of object is a class; clases are first-class values in Python, can be passed as parameters
- Everything in python is an object

Mutation Operations

```python
>>> suites = ['coin', 'string', 'myriad']
>>> suits.pop()
'myriad'
>>> suits.pop(0)
'coin'
>>> suits.remove('string')  #return None, no return
>>> suits.extend(['sword', 'club'])
```

Tuples

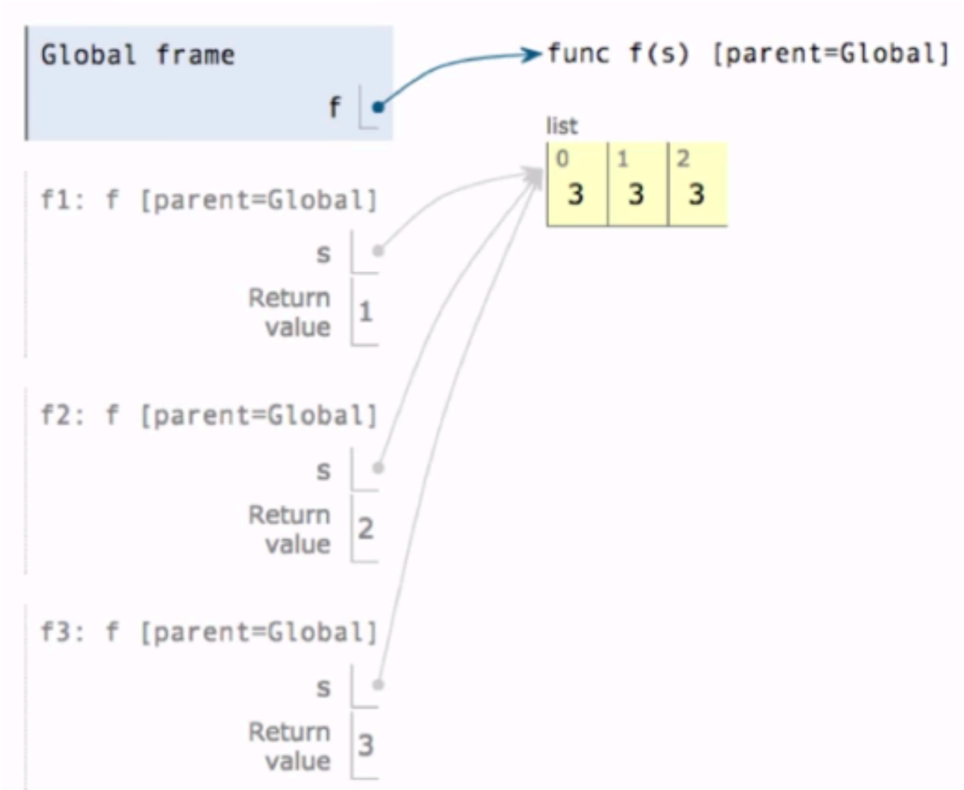
```python
>>> (3, 4) + (5, 6)
(3, 4, 5, 6)

>>> {(3, 4):5}  #a tuple can be a key
>>> {(3, [4, 6]):5}  #wrong a tuple can't be a key if there's any list inside

>>> a = ([1,2], 4)
>>> a[0].append(3)
>>> a
([1,2,3], 4)
```

Mutation

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

>>> a = [10]
>>> b = [10]
>>> a == b
True
#identity operator
>>> a is b
False
```

Mutable Default Arguments are Dangerous

```python
>>> def f(s=[]):
...     s.append(5)
...     return len(s)
>>> f()
1
>>> f()
2
>>> f()
3
```



[return to the top](#)

Program Decompositio and Debugging

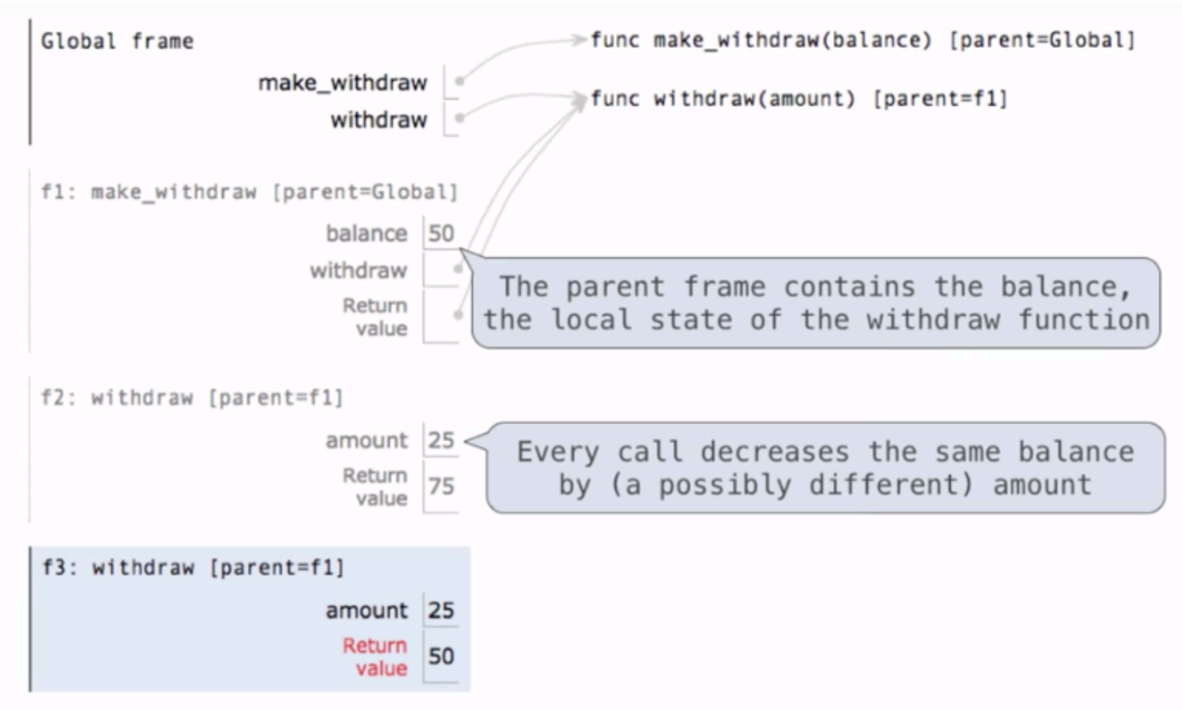
- Function Rules in Practice
  - solve one problem
  - smallest number of parameters
  - repeated sequence should be put in its own function

[return to the top](#)

Multable Functions and None Local

Python

```
>>> withdraw = make_withdraw(100)
>>> withdraw(25)
75
>>> withdraw(25)
50
```

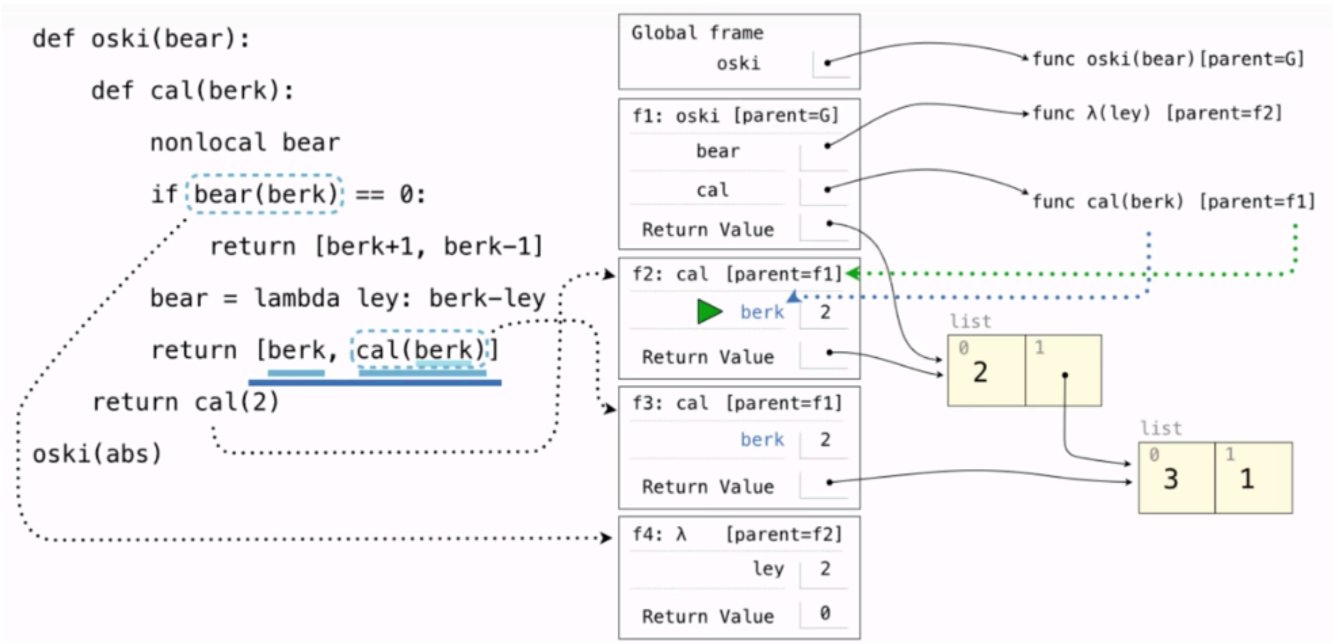



```python

def make_withdraw(balance):
    def withdraw(amount):
        nonlocal balance
        if amount > balance:
            return 'Insufficient funds'
        balance = balance - amount
        return balance
    return withdraw
```

- balance is rebind in the first non-local frame (enclosing scope in python doc)
- The name must not collide with pre-existing bindings in the local scope
- If there's not 'balance' exist in the upper layers of frames, it will cause an error
- If there's a local binding for 'balance', it will also cause an error
- a variable defined in the global frame can not be declared non-local, must be used in higher order functions

> python particularly pre-computes wich frame contains each name before executing the body of a function.
>
> Within the body of a function, all instances of a name must refer to the same frame

```python
#alternative
def make_withdraw_list(balance):
    b = [balance]
    def ...
```



[return to the top](#)

Iterators

```python
"""
iter(iterable)
next(iterator)
"""
>>> s = [3, 4, 5]
>>> t = iter(s)
>>> next(t)
3
>>> next(t)
4
>>> next(t)
5
>>> next(t)
StopIteration Error    #end if go outside the list
```

Iterate through dictionary

- keys, values, items of a dictionary are all iterables, the order of items is the order they were added(python 3.6+)
- if the shape or structure of the dictionary is changed while being iterated, there will be an error.(It's ok to change the values)
- The modification of the list will affect the result of showing through an iterator

```python
>>> i = iter(d.iterms())
>>> next(i)
('one', 1)
```

Iterator and For

- Iterator will be moved by for

```python
>>> r = range(3, 6)
>>> it = iter(r)
>>> next(it)
3
>>> for i in it:
...     print(i)
...
4
5
>>> for i in it:
...     print(i)
...
#Nothing because it already reached the end of the iterable
```

- Built-in functions for Iteration

```python
#func here will apply lazily (when we ask for the ith value
#they all return iterators

map(func, iterable)     #Iterate over func(x) for x in iterable, return map object
filter(func, iterable)  #Iterate over x in iterable if func(x)
zip(first_iter, second_iter)    #Iterate over co-indexed (x, y) pairs
reversed(sequence)      #Iterate over x in a sequence in reverse order

list(iterable)
tuple(iterable)
sorted(iterable)    #Create a sorted list containing x in iterable

>>> bcd = ['b', 'c', 'd']
>>> [x.upper() for x in bcd]

>>> list(iterator)  #create a list using iterator
```

Generators

```python
#yield all the paths that reach the value x
#yield will only 1 layer a time, it won't cause duplicated sub-paths
def generate_paths(t, x):
    if t.label == x:
        yield [t.label]
    for b in t.branches:
       for path in generate_paths(b, x):
           if path:
               yield [t.label] + path
```

```
>>> def plus_minus(x):
...     yield x
...     yield -x

>>> t = plus_minux(3)
>>> next(t)
3
>>> next(t)
-3
>>> t
<generator object ....>

def evens(start, end):
    even = start + (start %2)
    while even < end:
        yield even
        even += 2

>>> list(evens(1, 10))
[2, 4, 6, 8]
>>> t = evens(2, 10)
>>> next(t)
2


>>> list(a_then_b[3, 4], [5, 6]))
[3, 4, 5, 6]

def a_then_b(a, b):
    yield from a
    yield from b



>>> list(countdown(5))
[5, 4, 3, 2, 1]

def countdown(k):
    if k > 0:
        yield k
        yield from countdown(k-1)


def countdown(k):
    if k > 0:
        yield k
        yield countdown(k-1)
>>> t = countdown(3)
>>> next(t)
```

```
3
>>> next(t)
<generator object countdown ....>


def prefixes(s):
    if s:
        yield from prefixes(s[:-1])
        yield s
>>> list(prefixed('both'))
['b', 'bo', 'bot', 'both']

def substrings(s):
    if s:
        yield from prefixes(s)
        yield from substrings(s[1:])
>>> list(substrings('tops')
t', 'to', 'top', 'tops', 'o', 'op', 'ops', 'p', 'ps', 's']
```

return to the top

Growth

Python

```
total = 0
def count(f):
    def counted_f(*args):
        global total
        total += 1
        return f(*args)
    return counted_f

def fact(n):
    if n<=1:
        return 1
    return n * fact(n-1)

>>> fact = count(fact)
>>> fact(10)
xxxxx
>>> total
10
```

return to the top

Object Oriented

- Method calls are messages passed between objects
- A class statement creates a new class and binds that class to in the first frame of the current environment

```python
#find in instance, then class
>>> getattr(tom_account, 'balance')
10

>>> hasattr(tom_account, 'deposit')
True

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

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

Attributes

- All objects have attributes, which are name-value pairs
- Methods are also attributes of the class

```python
class Account:
    interest = 0.02
    def __init__(self, holder):
        self.holder = holder
        self.balance = 0

#If the attribute of the instance doesn't exist, it will create one
>>> tom_account.interest = 0.08
>>> tom_account.interest
0.08

#use parent class method
Account.withdraw(self, ~)

>>> self.withdraw_fee #This will evaluated to the class attribute if there's no one for the inst
>>> self.withdraw_fee += 10 #This will also evaluated to the class attribute if there's no one f


class Dog:
    def bark(self):
        print('woof!')

>>> lacey = Dog()
>>> lacey.bark = Dog.bark

>>> lacey.bark()
Error  #need an arguement self

#kind is a class name
def get_object(kind):
    return kind()
```

Composition

- One object hold another one as an attribute

```python
class B:
    n = 4
    def __init__(self, y):
        self.z = self.f(y)

class C(B):
    def f(self, x):
        return x

#Even if it calls the parent's method, the self is still represent itself
>>> C(2).z
2
```

return to the top

Linked Lists

```python
isinstance(rest, Link) #to see whether rest is a Link
```

Property Methods

- They are called implicitly

```python
class Link:
    @property
    def second(self):
        return self.rest.first

    @second.setter
    def second(self, value):
        self.rest.first = value

#[3, 4, 5]
>>> s.second
4
>>> s.second = 5
>>> s.second
5
```

return to the top

Magic Methods

Python

```python
class A:

    def __str__(self):
        return 'A object'


>>> print(A())
A object


class A:
    def __repr__(self):
        return 'A object'

>>> a = A()
#str default use repr
>>> print(a)
A object
>>> a
A object
```

Python

```python
#Full linked list
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert type(rest) is Link or \
        rest is Link.empty, \
        'rest must be a linked list or empty'
        self.first = first
        self.reset = rest

    def __repr__(self):
        if self.reset is Link.empty:
            return 'Link(' + repr(self.first) + ')'
        return 'Link(' + repr(self.first) + ', ' + repr(self.rest) + ')'

    def __str__(self):
        s = '<'
        while self.rest is not Link.empty:
            s += str(self.first) + ', '
            self = self.rest
        return s + str(self.first) + '>'

    def __contains__(self, elem):
        if self.first == elem:
            return True
        elif self.rest is Link.empty:
            return False
        return elem in self.rest

```

```
    def __add__(self, other):
        if self.rest is Link.empty:
            if other.rest if Link.empty:
                return link(self.first, Link(other.first))
            else:
                return Link(self.first, Link(other.first) + other.rest)
        else:
            return Link(self.first, self.rest + other)

    #l*2
    def __mul__(self. other):
        temp = self
        for _ in range(other - 1):
            temp = temp + self
        return temp

    #2*l
    def __rmul__(self, other):
        return self * other

    #len(l)
    def __len__(self):
        return 1 + len(self.rest)

    #l[0], max(l), min(l)
    def __getitem__(self, index):
        if type(index) is int:
            if index == 0:
                return self.first
            return self.rest[index - 1]
        #for slicing [1:3]
        elif type(index) is slice:
            start = slice.start or 0 #None then 0
            stop = slice.stop or len(self) #None then len(self)
            steop = index.step or 1

            if stop <= start:
                return Link.empty
            if start == 0:
                return Link(self.first, self[start + step:stop:step])
            return self.rest[start - 1:stop - 1:step]

#start 1 stop 2 steps 3
>>> slice(1, 2, 3)

>>> max(l)
3
>>> min(l)
1

>>> l = Link(1, Link(2, Link(3)))
>>> l
```

```
Link(1, Link(2, Link(3)))

>>> print(l)
<1, 2, 3>

>>> 3 in l
True
>>> 5 in l
False

>>> l2 = Link(1, Link(3))

>>> l + l2
Link(1, Link(2, Link(3, Link(1, Link(3)))))
```

[return to the top](#)

Error Handling

Python

```python
try:
    1 + 'hello'
except NameError as e:
    print('Error message:', e)
```

[return to the top](#)

Miscellaneous

week2 miscellaneous

- we aren't normally allowed to modify variables defined in parent frames
  `python def parent(previous_val): def child(): previous_val += 1 #not allowed, will cause error`

week3 miscellaneous

```python
>>> party = [1, 2]
>>> rival = party
>>> party = party + [4]
>>> rival
1, 2

>>> random.choice([1,2,3]) #randomly choose an element

>>> a = [1, 2]
>>> b = [3, 4]
>>> c = zip(a, b)   #c is an object
>>> list(c)
[(1, 3), (2, 4)]

>>> a = [1, 2]
>>> b = [3]
>>> list(zip(a, b))
[(1, 3)]
```

> Which of the following operations breaks the abstraction barrier?
>
> a. branches(t)[0]
>
> b. label(t)
>
> c. label(branches(t)[0])
>
> d. t[0]
>
> e. branches(t)[0][1]
>
> d e
>
> Which of the following are necessary qualities of a function that does not need to be broken into smaller functions?
>
> a. The function is called in multiple other parts of the program.
>
> b. The function solves one problem.
>
> c. The function does not contain repeated sequences of code. The function is recursive.
>
> e. A subset of the body of the function contains logic that could be re-used in another program.
>
> b c

Midterm Miscellaneous

```python
>>> 1==True
True
>>> 0==False
True
>>> 2==True
False
>>> 2==False
False
>>> list(a) is a
False

>>> def f():
...     return 'test'
>>> f()
'test'

>>> sum([1, 2, 3], 5)
11
>>> sum([1, 2, 3], [3])
Error
>>> sum([[1, 2, 3]], [4])
[1, 2, 3, 4]

#==, != only return True False, but and, or returns last possible value evaluated
>>> True==1
True
>>> True and 1
1
```

List

```python
append(obj)->None
count(val)->int
extend(iterable)->None

index(val, start=0, stop=9~)->:
- first index of val
- Value Error if not exist

insert(index, object)->None

pop(index=-1)->:
- iterm removed
- Index Error if not found

remove(val)->: (remove first occurance)
- None
- Error if not found

reverse()->None

sort(key=None, reverse=False)->None (default asc)
```

Dictionary

```python
get(key, default=None)->
items()iterable->iterable->tuples
keys()->iterable inside
pop(key, [d])->
- val
- d if not found
- error if no d and not found
update(dict)->None
values()->iterable
```

str

```python
index(sub, [start], [end])->:
- int
- Error if not found

find(sub, [start], [end])->:
- int
- -1

replace(old, new, count=-1)->copy of str (-1 means all)

'{0} is a good {1}'.format('ly', 'boy)
```

Week5 Miscellaneous

Python
```python
float('inf')   #inifity
```

return to the top

Scheme

Expressions

Call Expressions

Scheme
```scheme

> quotient
#[quotient]
> 'quotient
quotient
> /
#[/]

;operator in the parenthesis
> (quotient 10 2)
5

> (quotient (+ 8 7) 5)
3

> (quotient 1 2)
0
> (/ 1 2)
0.5

;change line anywhere
> (+ (* 3
          (+ (* 2 4)
              (+ 3 5)))
      (+ (- 10 7)
          6))
57

//special cases
scm> (+)
0
scm> (*)
1
scm> (* 2 2 2)
8
scm> +
```

```
#[+]

;number? is a name
scm> (number? 3)
#t
scm> (number? +)
#f
scm> (zero? 2)
#f
scm> (zero? 0)
#t
scm> (integer? 2)
#t

> (modulo 35 4)
3

> (even? 2)
#t
> (odd? 2)
#f

> (not (= 1 2))
#t

> (eq? 1 2)
#f
> (= `a `b)
Error
> (eq? `a `b)
#f
> (equal? `a `a)
#t

> (pair? (cons 2 nil))
#t
```

Special Forms

- A combination that is not a call expression

Scheme

```scheme
- If expression: (if <predicate> <consequent> <alternative>)
- And and or: (and <e1> ... <en>), (or <e1> ... <en>)
- Binding symbols: (define <symbol> <expression>)
- New procedres: (define (<symbol> <formal
- parameters>) <doby>)

> (define pi 3.14)
> (* pi 2)
6.28

> (define (abs x)
    (if (< x 0)
        (- x)
        (x)))

>(abs -3)
>3

> (define (average x y)
    (/ (+ x y) 2))
> (average 3 7)
5

> (let ((a 1)) a)
1
> (let ((a 1)(b a)) b)
Error
```

Recursion

```scheme
scm> (define (sqrt x)
        (define (update guess)
            (if (= (square guess) x)
                guess
                (update (average guess (/ x guess)))))
        (update 1))
sqrt
scm> (sqrt 256)
16


(define (mystery lst)
    (cond
        ((null? lst) #f)
        ((eq? (car lst) 61) #t)
        (else (mystery (cdr lst)))
    )
)

> (let (v 1) (b 2) (v+b))
3
```

(cdr `(10 21))

lambda Expressions

```scheme
lambda (<formal-parameters>) <body>
;Same here
(define (plus4 x) (+ x 4))
(define plus4 (lambda (x) (+ x 4)))

((lambda (x y z) (+ x y (squarez))) 1 2 3)
```

Pairs and Lists

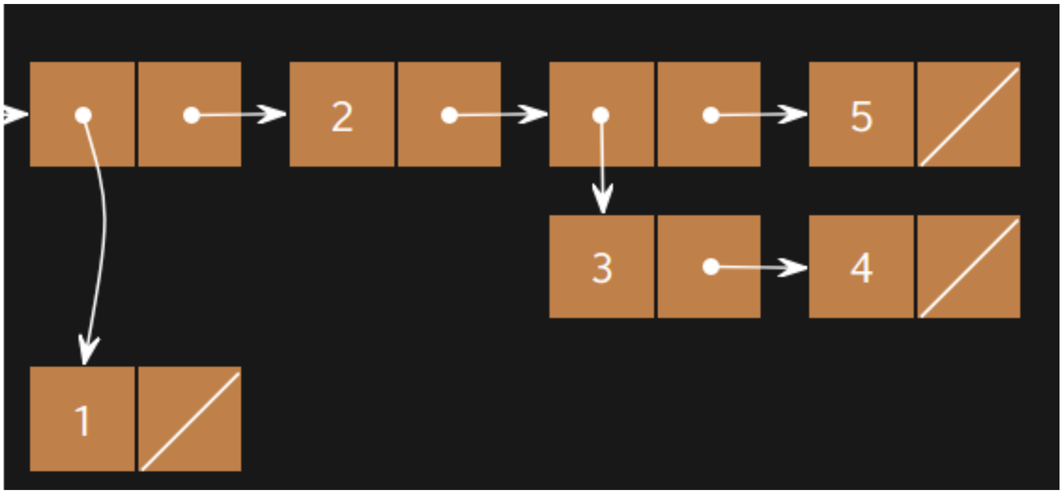
Pairs

```scheme
> (cons 1 2)
;car return the first in the pair
;cdr return the second
;nil the empty list
```

List

Scheme

```
;The last element must be nill
> (cons 1 (cons 2 nill)) ;2 elements list
(1 2)

> (define a 1)
> (define b 2)
> (list a b)
(1 2)

> (list 'a 'b)
(a b)

> (car '(a b c))
a
> (cdr '(a b c))
(b c)

> (length `(1 2 3 4 5))
5

> (append `(1 2 3) `(4 5 6))
(1 2 3 4 5 6)

> (cdr (cons 1 (cons 2 nil)))
(2)
> (car (cons 1 (cons 2 nil)))
1

> (cons 1 `(list 2 3))
(1 list 2 3)

> (define l (cons 4 (cons 3 (cons 2 nil))))
> (append l '(1 0))
(4 3 2 1 0)

;Tricky part
> (define a `(1))
> (define b (cons 2 a))
> b
(2 1)
> (define c (list 3 b))
(3 (2 1))
> (cdr c)
((2 1))
```



- The first is val, the second is pointer - next node is at the same level - If the first is a pointer, there'll be a sub list

```Scheme
> (cons (cons 1 nil) (cons 2 (cons (cons 3 (cons 4 nil)) (cons 5 nil))))
((1) 2 (3 4) 5)

> (list (cons 1 nil) (cons 2 (cons (cons 3 (cons 4 nil)) (cons 5 nil))))
((1 ()) 2 (3 4) 5)
```

Note: null? is a symbol to verify if the thing behind it is nil or not. There's no 'null'. 'nil' is just () empty list

Dynamic Scope and Lexical scope

```Scheme
> (define f (lambda (x) (+ x y)))

> (define g (lambda (x y) (f (+ x x))))

>(g 3 7)
```

- Lexical scope: The parent for f is the global (will cause error, no y)
- Dynamic scope: The parent for f is g

Functional Programming

- All functions are pure functions
- No re-assignment and no mutable data types
- Name-value bindings are permanent
- Advantages
    - The value of an expression is independent of the order in which sub-expressions are evaluated
    - Sub-expressions can safely be evaluated in parallel or on demand (lazily)
    - Referential transparency: The value of an expression does not change when we substitute one of its subexpression with the value of that subexpression.

But... no for/while statements, how to iteration efficient? Tail Recursion

Tail Recursion

A precedure call that has not yet returned is active. Some procedure calls are tail calls. A Scheme interpreter should support an unbounded number of active tail calls using only a constant amount of space.

A tail call is a call expression in a tail context:

- The last body sub-expression in a lambda expression
- Sub-expressions 2 & 3 in a tail context if expression
- All non-predicate sub-expressions in a tail context cond
- The last sub-expression in a tail context and or or
- The last sub-expression in a tail context begin

```
(define (factorial n k)
  (if (= n 0) k
    (factorial (- n 1)
               (* k n)) ) )
```

- A call expression is not a tail call if more computation is still required in the calling procedure.
- Linear recursive procudures can often be re-written to use tail calls.

```
(define (length s)
  (if (null? s) 0
    (+ 1 (length (cdr s)) ) ) )
```

Not a tail context

```
(define (length-tail s)
  (define (length-iter s n)
    (if (null? s) n
      (length-iter (cdr s) (+ 1 n)) ) )
  (length-iter s 0) )
```

More Examples

```scheme
;; Compute the length of s.
(define (length s)
  (+ 1 (if (null? s)
          -1
          (length (cdr s))) ) )
```

Not a tail recursion call

Map and Reduce

Reduce

Scheme

```scheme
(define (reduce procedure s start)
    (if (null? s) start
        (reduce procudure
                (cdr s)
                (procedure start (car s)))))
;it depends on the procedure, if it's a constant space produce it's constant space

> (reduce * `(3 4 5) 2)     ;2 * 3 * 4 *5
120
> (reduce (lambda (x y) (cons y x)) `(3 4 5) `(2))
(5 4 3 2)
```

Map

```scheme
;Not a tail recursion version
(define (map procedure s)
    (if (null? s)
        nil
        (cons (procedure car s))
                (map procedure (cdr s)))))

exp:
(map (lambda (x) (- 5 x)) (list 1 2))

(define (map procedure s)
    (define (map-reverse s m)
        (if (null? s)
            m
            (map-reverse (cdr s)
                            (cons (procedure (car s))
                                    m))))
    (reverse (map-reverse s nill)))

(define (reverse s)
    (define (reverse-iter s r)
        (if (null? s)
            r
            (reverse-iter (cdr s)
                             ( cons (car s) r))))
        (reverse_iter s nill))
```

Filter

```scheme
(define (unique s)
  (if (null? s)
      nil
      (cons (car s)
            (unique (filter (lambda (x) (not (eq? x (car s))))
                            (cdr s)
                    )
            )
      )
  )
)
```

append

```scheme
scm> (append '(1 2 3) '(4 5 6))
(1 2 3 4 5 6)
scm> (append)
()
scm> (append '(1 2 3) '(a b c) '(foo bar baz))
(1 2 3 a b c foo bar baz)
scm> (append '(1 2 3) 4)
Error
```

Macro

```scheme
Primitive: 2 3 true + quotient
Combinations: (quotient 10 2) (not true)

> (list 'quotient 10 2)
(quotient 10 2)

> (eval (list 'quotient 10 2))
5

> (list + 1 2)
(#[+] 1 2)

> (list '+ 1 2)
(+ 1 2)

> (list '+ (+ 2 3))
(+ 5)

> (define (fact-exp n)
    (if (= n 0) 1 (list '* n (fact-exp(- n 1)))))
> (fact-exp 5)
(* 5 (* 4 (* 3 (* 2(* 1 1)))))

> (eval (fact-exp 5))
120

> (define (fib-exp x)
    (if (<= n 1) n (list '+ (fib-exp (- n 2)) (fib-exp (- n 1)))))

> (fib-exp 4)
(+ (+ 1 (+ 0 1)) (+ (+ 0 1) (+ 1 (+ 0 1))))
```

- A macro is an operation performed on the source code of a program before evaluation
- Evaluate the operator, if it evaluates to a macro call the macro on the source code (eval the source code and replace the user input as string into the source code unless there's a comma)
- Then evaluate the expression returned from the macro procedure

Scheme

```scheme
> (define-macro (twice expr)
    (list 'begin expr expr))

> (print 2)
2

> (begin (print 2) (print 2))
2
2

> (define (twice expr) (list 'begin expr expr))
> (twice (print 2))
(begin None None)
> (twice '(print 2)) ;' stop it from evaluating
(begin (print 2) (print 2))
> (eval (twice '(print 2)))
2
2
> (defin-macro (twice expr) (list 'begin expr expr))
> (twice (print 2))
2
2

> (define (check val) (if val 'passed 'failed))
> (define-macro (check expr)(list 'if expr ''passed ''failed)
> (define x -2)
x
> (check (> x 0))
failed


> (define-macro (check expr)(list 'if expr ''passed
    (list 'quote (list 'failed: expr))))
> (check (> x -2))
(failed: (> x -2))

None is true
;for macro
> (define (map fn vals)
    (if (null? vals)
        ()
        (cons (fn (car vals))
                (map fn (cdr vals)))))

> (define-macro (for sym vals expr)
    (list 'map (list 'lambda (list sym) expr) vals)
> (for x '(2 3 4 5) (* x x))
(4 9 16 25)

```

Quasi-Quotation

- parts of it can be evaluate

```
Scheme
> (define b 2)
> '(a b c)
(a b c)
> `(a b c)
(a b c)

> `(a ,b c) ;if b can't be evaluate, there'll be an error
(a 2 c)

> '(a ,b c)
(a (unquote b) c)

> (define expr '(* x x))
> `(lambda (x) ,expr)
(lambda (x) (* x x))

scm> (define-macro (f x) (car x))
scm> (f (+ 2 3))
#[+]

scm> (f (quote (1 2))
Error
;anything will be evaluate to a val finally and quote can't be evaluate to a val alone
scm> quote
Error

scm> +
#[+]

scm> (define quote 7000)
scm> (f (quote (1 2))
7000

scm> '(1 ,x 3)
(1 (unquote x) 3)
```

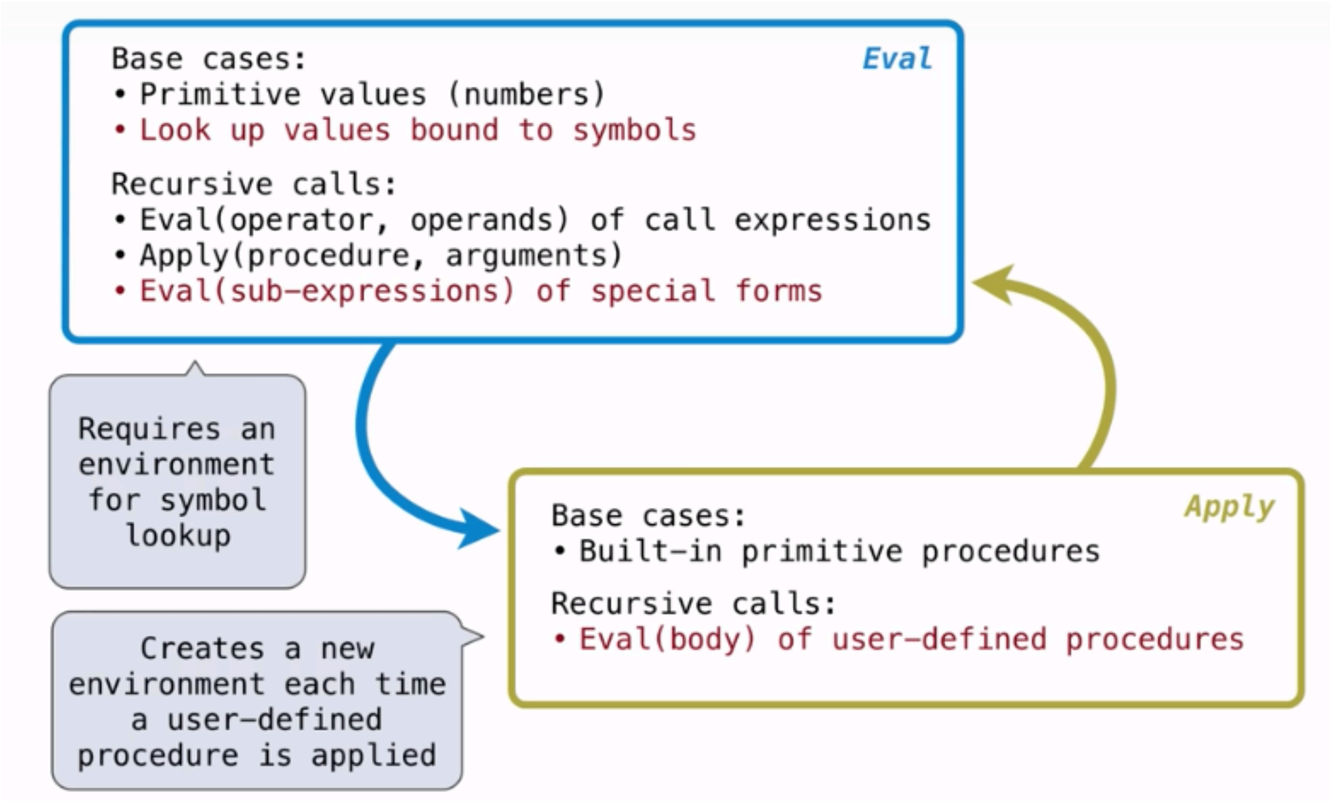
- symplify `scheme (define-macro (check expr) `(if ,expr 'passed '(failed: ,expr)))`

Scheme

```
;for loop
(define (cddr s) (cdr (cdr s)))
(define-macro
 (list-of map-expr for var in lst (variadic y))
 (list 'map
       (list 'lambda (list var) map-expr)
       (if (null? y)
           lst
           `(filter (lambda (,var) ,(cadr y)) ,lst)
       )
 )
)

; ; List all ways to make change for TOTAL with DENOMS
(define (list-change total denoms)
  (define (l-change total denoms path)
    (cond
      ((null? denoms)
       nil
      )
      ((< total (car denoms))
       (l-change total (cdr denoms) path)
      )
      ((> total (car denoms))
       (append (l-change (- total (car denoms))
                         denoms
                         (append path (list (car denoms))))
               )
               (l-change total (cdr denoms) path)
       )
      )
      ((= total (car denoms))
       (append (list (append path (list (car denoms))))
               (l-change total (cdr denoms) path)
       )
      )
    )
  )
  (l-change total denoms nil)
)
```

Back to Top

Interpreter



Special Forms

- Symbols are bound to values in the current environment
- else-evaluating expressions are returned.
- All other legal expressions are represented as Scheme lists, called combinations

Scheme

```scheme
;high order function
(define (outer x y)
    (define (inner z x)
        (+ x (* y 2) (* z 3)))
    inner)

> (outer 1 2)
inner

> ((outer 1 2) 1 10)
17
```

Logical Special Forms

- May evaluate only part of it
- The interpreter convert ' to (quote ~)

```scheme
> (if #t 1 (/ 1 0)) ;No error at all

> (and 0 1 nil #f 2)
;evaluated 0 1 nil #f, 0 and nill are all true

> (quote (+ 1 2)) ;or '(+ 1 2)
(+ 1 2)

> `(1 2)
(1 2)
```

Lambda Expressions

- Use a class

Frames and Environments

- Frames have parents(env)
- Frames are Python instances with methods lookup and define
- Lookup is a function recursively lookup from child to parent

Define Expressions

- binds a symbol to a value in the first frame of the current environment

- Procedure definition is shorthand of define with a lambda expression.

```scheme
(define (<name> <formal parameters>) <body>)
(define <name> (lambda (<formal parameters>) <body>))
```

Applying User-Defined Procedures

- To apply a user-defined procedure, create a new frame in which formal parameters are bound to argument values, whosw parent is the env of the procedure.
- Evaluate the body of the procedure in the environment that starts with this new frames.

```scheme
> (lambda (x)(+ x 6))
(lambda (x)(+ x 6))
```

return to the top

Streams

```python
def sum_primes(a, b):
    return sum(filter(is_prime, range(a, b)))

def is_prime(x):
    if x <= 1:
        return False
    return all(map(lambda y: x % y, range(2, x)))
```

- Space $\theta(1)$ (benifit by generator)

```scheme
(define (range a b)
    (if (>= a b) nil (cons a (range (+ a 1) b))))

(define (filter f s)
    (if (null? s)
        nil
        (if (f (car s))
            (cons (car s)
                (filter f (cdr s)))
            (filter f (cdr s)))))

(define (reduce f s start)
    (if (null? s)
        start
        (reduce f
                  (cdr s)
                  (f start (car s)))))

(define (sum s)
    (reduce + s 0))

(define (prime? x)
    (if (<= x 1)
        false
        (null? (filter (lambda (y) (= 0 (remainder x y))) (range 2 x)))))

(define (sum-primes a b)
    (sum (filter prime? (range a b))))
```

- Space $\theta(n)$

Solution

```scheme
;only evaluate 2 when cdr-stream is called
(cdr-stream (cons-stream 1 2)) -> 2

(cons-stream 1 (cons-stream 2 nil)

(cons-stream 1 (/ 1 0)) -> (1 . #[delayed]) ; No error
(cdr-stream (cons-stream 1 (/ 1 0)) ;error
```

Build Stream

```scheme
(define (range-stream a b)
    (if (>= a b) nil (cons-stream a (range-stream (+ a 1) b))))kjlk

;Infinite stream
(define (int-stream start)
    (cons-stream start (int-stream (+ 1 start))))

(define (square-stream s)
    (cons-stream (* (car s)(car s))
                (square-stream (cdr-stream s))))

(define ones (cons-stream 1 ones))

(define (add-streams s t)
    (cons-stream (+ (car s) (car t))
                (add-streams (cdr-stream s)
                                (cdr-stream t))))

;1 2 3 4 5....
(define ints (cons-stream 1 (add-streams ones ints)))

;map filter reduce stream
(define (map-stream f s)
    (if (null? s)
         nil
         (cons-stream (f (car s))
                        (map-stream f
                           (cdr-stream s)))))

(define (reduce-stream f s start)
    (if (null? s)
        start
        (reduce-stream f
                 (cdr-stream s)
                 (f start (car s)))))

(define (filter-stream f s)
    (if (null? s)
        nil
```

```
        (if (f (car s))
            (cons-stream (car s)
                (filter-stream f (cdr-stream s)))
            (filter-stream f (cdr-stream s)))))

;stream of primes
;filter all the multiple of 1 , 2, 3 until n
(define (sieve s)
    (cons-stream (car s)
                (sieve (filter-stream
                    (lambda (x) (not (= 0 (remainder x (car s)))))
                    (cdr-stream s)))))

define primes (sieve (int-stream 2)))
```

Promise

- A promis is an expressions, along with an environment in which to evaluate it
- lexical scope
- Delaying an expression creates a promis to evaluate it later in the current environment

Scheme

```
scm> (define promise (let ((x 2)) (delay (+ x 1))))
scm> (define x 5)
scm> (force promise)
3
```

- Every time writing delay, it just like create a lambda with no arguments

Scheme

```
(define-macro (delay expr) `(lambda () ,expr))
(define (force promise) (promise))

(define-macro (cons-stream a b) `(cons ,a (delay, b)))
(define (cdr-stream s) (force (cdr s))) //evaluate the lambda

scm> (define ones (cons-stream 1 ones))
(1 . #[promise (not forced)])
; not forced means hasn't been evaluated, if it has been evaluated, it will store the value and
```

Exp WWSD

```scheme
scm> (define oski 61)
oski
scm> (define go-bears (cons-stream oski (cons-stream oski nil)))
go-bears
scm> (define oski 1866)
oski
scm> (car (cdr-stream go-bears))
(1866)
```

return to the top

Declarative Language

- A "program" is a description of the desired result
- The interpreter figures out how to generate the result
- python is a imperative language
  - A "program" is a description of computational processes
  - The interpreter carries out execution/evaluation rules

SQL

```bash
sqlite3 -init ex.sql
sqlite>
```

SELECT

- select statement is used to create table

```sql
//create new permanent table
create table cities as
    select 38 as latitude, 122 as longitude, "berkeley" as name union
    select 42, 71, "Cambridge";


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

- arithmatic

```sql
select chair, single + 2 * couple as total from lift;
select word, one+two+four+eight as value from ints where one + two/2 + four/4 + eight/8 = 1;
```

Joining Two Tables

```sql
//join table using child = name
select parent from parents, dogs where child = name and fur = "curly";
```

Aliases and Dot Expressions

```sql
select a.child as first, b.child as second
    from parents as a, parents as b
    where a.parent = b. parent and a.child < b.child
```

Numerical and String Expressions

```sql
//<> != are the same
sqlite> select "hello," || " world";
hello, world

//substr, instr(position) not very good low efficiency
sqlite> select substr(s, 4, 2) || substr(s, instr(s, " ")+1, 1) from phrase;
low

//not good
sqlite> create table lists as select "one" as car, "two, three, four" as cdr;
sqlite> select substr(cdr, 1, instr(cdr, ",")-1) as cadr from lists;
two
```

Aggregate Functions

```sql
create table animals as
  select "dog" as kind, 4 as legs, 20 as weight union
  select "cat"         , 4        , 10          union
  select "ferret"      , 4        , 10          union
  select "parrot"      , 2        , 6           union
  select "penguin"     , 2        , 10          union
  select "t-rex"       , 2        , 12000;
```

```sql
/*max min ... will only display the max/min row*/
select max(legs) from animals;

--support operator
select max(legs-weight) + 5 from animals;

select min(legs), max(weight) from animals where name <> "t-rex"
3/6

--count(*) count number of rows, the following have the same results
select count(legs) from animals;
select count(kind) from animals;
select count(*) from animals;

--1 for each type, 2types
select count(distinct legs) from animals;
2

select count(distinct weight) from animals;
4

select sum(distinct weight) from animals;
12036 --ignore the 2 redundent "10"
```

- An aggregate function also selects a row in the table, which may be meaningful

```sql
select max(weight), kind from animals; -- we get only 1 row
12000|t-rex

select avg(weight), kind from animals;
2009.3333333333|t-rex --t-rex is not meaningful

select max(legs), kind from animals;
4|cat --There're 3 maxs 'cat' is not meaningful

select kind from animals where weight > 10 and weight = min(weight)
```

- group by
    - partition the rows in the table by group

SQL

```sql
select legs, max(weight) from animals group by legs;

legs max(weight)
4       20
2       12000

-- group by the cartisian product of legs and weight
select legs, weight from animals group by legs, weight;
2|6
2|10
2|12000
2|10
2|20

select max(kind), weight/legs from animals group by weight/legs
ferret|2 --10/4 default is 2, use "weight/legs/1.0" to get float
parrot|3
penguin|5
t-rex|6000

select weight/legs, count(*) from animals gropu by weight/legs having count(*)>1;
5 2
2 2
-- having clause filter the groups to leave the ones we want
```