Introduction to Computer and Programming

Lecture 9

Yue Zhang
Westlake University

August 1, 2023

WestlakeNLP

Chapter 9.

Class and Object

WestlakeNLP

Overview

WestlakeNLP

Overview

WestlakeNLP

Method

```
>>> l=[1,2,3]
>>> m=[4,5,6]
>>> l.extend([4,5])
>>> l
[1, 2, 3, 4, 5]
>>> m
[4, 5, 6]
```

```
>>> s="This is a boat"
>>> l=s.split()
>>> l
['This', 'is', 'a', 'boat']
```

- type: list
  object (instance): *l*
- type: str
  object (instance): *s*

Methods are type-specific, their calls are instance-specific.

WestlakeNLP

Attribute

```
>>> c=1+2j
>>> c
(1+2j)
>>> type(c)
<class 'complex'>
>>> c.real
1.0
>>> c.imag
2.0
>>> c.conjugate()
(1-2j)
```

- type: complex
  object (instance): c

Attributes are instance-specific.

WestlakeNLP

Types and Instances

- Attributes and methods are interfaces.
- Types decide interfaces.
- Instances decide the values.
- **Custom** types
  - custom type – class
  - instance – object

WestlakeNLP

Classes and Instances

```
>>> class Account:
...     pass      # empty statement
...
>>> a=Account()  # new instance
>>> a
<__main__.Account object at 0x1027919d0>
>>> type(a)
<class '__main__.Account'>
```

WestlakeNLP

Attributes

```
>>> class Account:
...     pass
...
>>> a=Account()
>>> b=Account()
```

```
>>> a.name='John'
>>> b.name='Marina'
>>> a.balance=1000
>>> b.balance=0
```

```
>>> a.balance
1000
>>> b.name
'Marina'
```

```
>>> del a.name
>>> a.name
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Account' object has no
    attribute 'name'
```

- set attribute
- get attribute
- del attribute

WestlakeNLP

Methods

```
>>> class Account:
...     def deposit(self,amount):
...         self.balance += amount
...
>>> a=Account()
>>> a.balance=10000
>>> a.balance
10000
```

```
>>> a.deposit(1000)
>>> a.balance
11000
```

```
>>> Account.deposit(a,1000)
>>> a.balance
12000
```

```
>>> b=Account()
>>> b.deposit(1000)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in deposit
AttributeError: 'Account' object has no
    attribute 'balance'
```

- call *Account.deposit* passing *a* as *self*
- the navie explicit form
- not assigned

WestlakeNLP

Constructor

A special method called when a new instance is constructed.

```
>>> class Account:
...     def __init__(self,name,balance=0):
...         self.name=name
...         self.balance=balance
...     def deposit(self,amount):
...         self.balance+=amount
...
>>> a=Account('Tim')
>>> b=Account('Marina',2000)
>>> a.deposit(1000)
>>> a.balance
1000
>>> b.balance
2000
```

WestlakeNLP

Constructor

- Class attributes

```
>>> class Account:
...     total=0
...     def __init__(self,name,balance=0):
...         self.name=name
...         self.balance=balance
...         self.account_num=Account.total+1
...         Account.total+=1
...     def deposit(self,amount):
...         self.balance+=amount
...
>>> a=Account('Tim')
>>> b=Account('Marina',2000)
>>> a.account_num
1
>>> b.account_num
2
```

Execution

```
>>> class Account:
...     total=0
...     def __init__(self,name,balance=0):
...         self.name=name
...         self.balance=balance
...         self.account_num=Account.total+1
...         Account.total+=1
...     def deposit(self,amount):
...         self.balance+=amount
...
>>> a=Account('Tim')
>>> b=Account('Marina',2000)
>>> a.account_num
1
>>> b.account_num
2
```

- The execution of class statement:

  - execute the class body statement by statement.

  - attach variables as class attributes.

  - attach functions as class methods.

  - similar to module import to some extent.

Execution

```
>>> class Account:
...     total=0
...     def __init__(self,name,balance=0):
...         self.name=name
...         self.balance=balance
...         self.account_num=Account.total+1
...         Account.total+=1
...     def deposit(self,amount):
...         self.balance+=amount
...
>>> a=Account('Tim')
>>> b=Account('Marina',2000)
>>> a.account_num
1
>>> b.account_num
2
```

- Execution of instance construction:
  - call constructor
  - return new instance
- Constructors are automatically triggered.
  - Are there other special methods?

Special Methods

```
>>> class A:
...     pass
...
>>> class B:
...     def __str__(self):
...         return "I am a B instance"
...
```

```
>>> a=A()
>>> str(a)
'<__main__.A object at 0x102bae820>'
>>> print(a)
<__main__.A object at 0x102bae820>
```

```
>>> b=B()
>>> str(b)
'I am a B instance'
>>> print(b)
I am a B instance
```

- use str
- triggers __str__

WestlakeNLP

Special Methods – Other Special Methods

- built-in functions and many operators

Method	Trigger
__int__(self)	int(x)
__float__(self)	float(x)
__nonzero__(self)	bool(x)
__len__(self)	len(x)
__neg__(self)	− x
__abs__(self)	abs(x)

WestlakeNLP

Special Methods – Other Special Methods

- binary operators

Method	Trigger
__add__(self,other)	x + y
__sub__(self,other)	x - y
__mul__(self,other)	x * y
__div__(self,other)	x / y
__floordiv__(self,other)	x // y
__mod__(self,other)	x % y
__pow__(self,other)	x**y
__eq__(self,other)	x == y
__gt__(self,other)	x > y

WestlakeNLP

Special Methods – Other Special Methods

- more binary operators

Method	Trigger
__contains__(self,y)	y in x
__getitem__(self,i)	x[i]
__setitem__(self,i,y)	x[i] = y
__getslice__(self,b,e)	x[b:e]
__setslice__(self,b,e,y)	x[b:e] = y
__call__(self,args)	x(args)
__getattr__(self,name)	x.name
__setattr__(self,name,value)	x.name = value
__delattr__(self,name)	del x.name

WestlakeNLP

Special Methods – Example

```
>>> class Matrix:
...     def __init__(self,l=[]):
...         import copy
...         self.x = copy.deepcopy(l)
...     def __add__(self,other):
...         import copy
...         r = copy.deepcopy(self.x)
...         for i in range(len(r)):
...             for j in range(len(r[i])):
...                 r[i][j] += other.x[i][j]
...         return Matrix(r)
...     def __str__(self):
...         s=''
...         for i in range(len(self.x)):
...             for j in range(len(self.x[i])):
...                 s += str(self.x[i][j])
...                 s += '\t'
...             s += '\n'
...         return s
...
```

```
>>> m=Matrix
    ([[1,2,3],[4,5,6],[7,8,9]])
>>> n=Matrix
    ([[1,0,0],[0,1,0],[0,0,1]])
>>> print(m)
1   2   3
4   5   6
7   8   9

>>> print(n)
1   0   0
0   1   0
0   0   1

>>> y=m+n
>>> print(y)
2   2   3
4   6   6
7   8   10
```

Inheritance

account.py

```
class Account:
    total = 0
    def __init__(self,name,balance=0):
        self.name=name
        self.balance=balance
        self.account_num=Account.total+1
        Account.total+=1
    def deposit(self,amount):
        assert amount>0
        self.balance+=amount
    def withdraw(self,amount):
        assert amount>0
        if amount>self.balance:
            print("Error:insufficient fund.")
            return
        self.balance-=amount
```

Inheritance

```
>>> import account
>>> class CreditAccount(account.Account):
...     pass
...
>>> a=account.Account('Tim')
>>> b=CreditAccount('Jack')
>>> a
<account.Account object at 0x10269a160>
>>> type(a)
<class 'account.Account'>
>>> b
<__main__.CreditAccount object at 0x1027bff40>
>>> type(b)
<class '__main__.CreditAccount'>
>>> b.deposit(1000)
>>> b.account_num
2
>>> b.balance
1000
>>> b.withdraw(2000)
Error:insufficient fund.
```

Inheritance

account.py

```
class Account:
    total = 0
    def __init__(self,name,balance=0):
        self.name=name
        self.balance=balance
        self.account_num=Account.total+1
        Account.total+=1
    def deposit(self,amount):
        assert amount>0
        self.balance+=amount
    def withdraw(self,amount):
        assert amount>0
        if amount>self.balance:
            print("Error:insufficient fund.")
            return
        self.balance-=amount
```

credit_account.py

```
from account import Account

class CreditAccount(Account):
    def __init__(self,name,balance=0,limit=0):
        Account.__init__(self,name,balance)
        self.limit=limit
    def withdraw(self, amount):
        assert amount>0
        if amount>self.balance+self.limit:
            print("Error:insufficient limit.")
            return
        self.balance-=amount
```

WestlakeNLP

Inheritance

- overrding methods

```
>>> from account import Account
>>> from credit_account import CreditAccount
>>> a=Account('Jack')
>>> b=CreditAccount('Rose',limit=500)
>>> a.balance
0
>>> b.balance
0
>>> a.withdraw(100)
Error:insufficient fund.
>>> a.balance
0
>>> b.withdraw(100)
>>> b.balance
-100
```

WestlakeNLP

Object Oriented Programming

- Three important elements of object oriented programming:
  - encapsulation
    - attribute, method
  - inheritance
    - subclass
  - polymorphism
    - operator, overrding

WestlakeNLP

Overview

WestlakeNLP

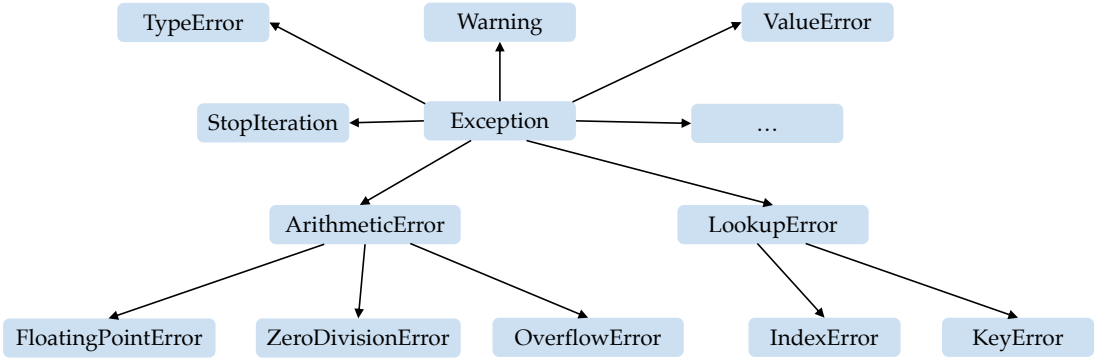
Exception Handling

- Exception: runtime error that prevents nomal control flow.
  - division by zero
  - type error
  - value error
  - index error

WestlakeNLP

Exception Handling

- Subclasses



WestlakeNLP

Exception Handling

- control flow

```
>>> def f(x):
...     print(1/x)
...     print('Done.')
...
>>> f(0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in f
ZeroDivisionError: division by zero
```

- control flow terminates

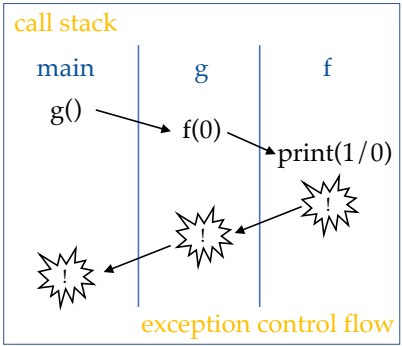
WestlakeNLP

Exception Handling

```
>>> def f(x):
...     print(1/x)
...     print('Done.')
...
>>> def g():
...     return f(0)+1
...
>>> g()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in g
  File "<stdin>", line 2, in f
ZeroDivisionError: division by zero
```

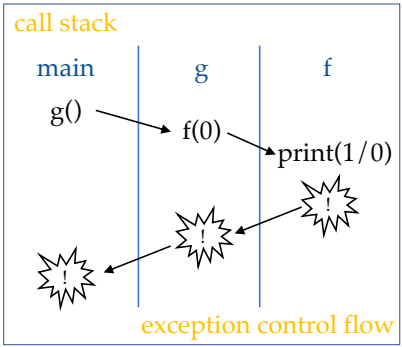
Exception Handling

```
>>> def f(x):
...     print(1/x)
...     print('Done.')
...
>>> def g():
...     return f(0)+1
...
>>> g()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in g
  File "<stdin>", line 2, in f
ZeroDivisionError: division by zero
```



Exception Handling

```
>>> def f(x):
...     print(1/x)
...     print('Done.')
...
>>> def g():
...     return f(0)+1
...
>>> g()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in g
  File "<stdin>", line 2, in f
ZeroDivisionError: division by zero
```



- can catch the exception in the exception control flow

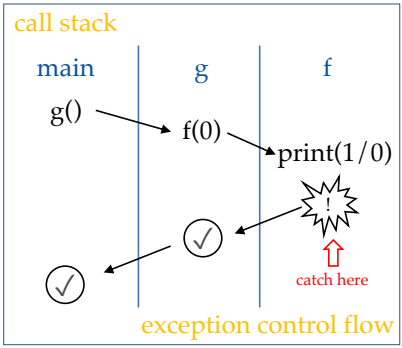
Exception Handling

```
>>> def f(x):
...     try:
...         print(1/x)
...         print('Done.')
...     except:
...         print('Error.')
...         return -1
...
>>> def g():
...     return f(0)+1
...
>>> g()
Error.
0
```

WestlakeNLP

Exception Handling

```
>>> def f(x):
...     try:
...         print(1/x)
...         print('Done.')
...     except:
...         print('Error.')
...         return -1
...
>>> def g():
...     return f(0)+1
...
>>> g()
Error.
0
```



WestlakeNLP

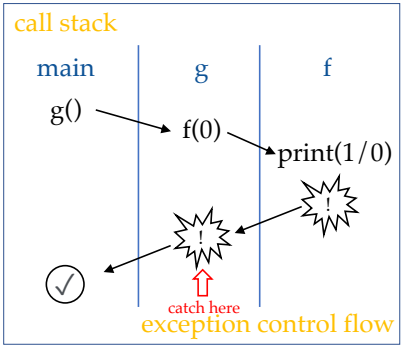
Exception Handling

```
>>> def f(x):
...     print(1/x)
...     print('Done.')
...     return f(0)+1
...
>>> def g():
...     try:
...         return f(0)+1
...     except:
...         return 0
...
>>> g()
0
```

WestlakeNLP

Exception Handling

```
>>> def f(x):
...     print(1/x)
...     print('Done.')
...     return f(0)+1
...
>>> def g():
...     try:
...         return f(0)+1
...     except:
...         return 0
...
>>> g()
0
```



WestlakeNLP

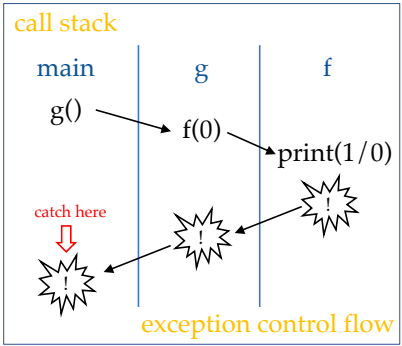
Exception Handling

```
>>> def f(x):
...     print(1/x)
...     print('Done.')
...     return f(0)+1
...
>>> def g():
...     return f(0)+1
...
>>> try:
...     g()
... except:
...     pass    # do nothing
...
```

WestlakeNLP

Exception Handling

```
>>> def f(x):
...     print(1/x)
...     print('Done.')
...     return f(0)+1
...
>>> def g():
...     return f(0)+1
...
>>> try:
...     g()
... except:
...     pass     # do nothing
...
```



WestlakeNLP

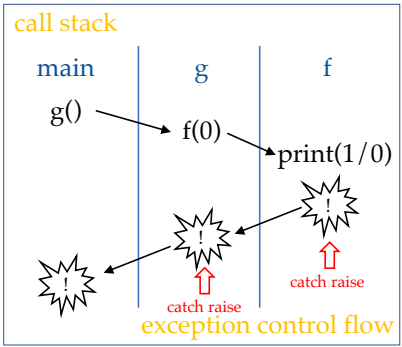
Exception Handling

```
>>> def f(x):
...     try:
...         print(1/x)
...         print('Done.')
...         return x
...     except ZeroDivisionError as e:
...         print('Error in f')
...         raise e
...
>>> def g():
...     try:
...         return f(0)
...     except Exception as e:
...         print('Error in g')
...         raise e
...
>>> g()
Error in f
Error in g
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in g
  File "<stdin>", line 3, in g
  File "<stdin>", line 8, in f
  File "<stdin>", line 3, in f
ZeroDivisionError: division by zero
```

WestlakeNLP

Exception Handling

```
>>> def f(x):
...     try:
...         print(1/x)
...         print('Done.')
...         return x
...     except ZeroDivisionError as e:
...         print('Error in f')
...         raise e
...
>>> def g():
...     try:
...         return f(0)
...     except Exception as e:
...         print('Error in g')
...         raise e
...
>>> g()
Error in f
Error in g
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in g
  File "<stdin>", line 3, in g
  File "<stdin>", line 8, in f
  File "<stdin>", line 3, in f
ZeroDivisionError: division by zero
```



WestlakeNLP

Exception Handling

- catching specific types of exceptions

```
>>> def f(x):
...     try:
...         return 1/x
...     except ZeroDivisionError:
...         print('zero division')
...         return -1
...     except (TypeError,ValueError):
...         print('type or value error')
...         return -2
...     except:
...         return -3
...
>>> f(0)
zero division
-1
>>> f('abc')
type or value error
-2
```

Exception Handling

- finally

```
>>> def f(x):
...     try:
...         return 1/x
...     except ZeroDivisionError:
...         print('zero division')
...         print(1/x)
...     except (TypeError,ValueError):
...         print('type or value error')
...         print(1/x)
...     except:
...         print(1/x)
...     finally:
...         print('Done')
...
>>> f(0)
zero division
Done
>>> f(1)
Done
1.0
```

Manipulating Files

- Text files

~/Desktop/abc.txt

```
abc
def
ghi
```

WestlakeNLP

Manipulating Files

- Text files

~/Desktop/abc.txt

```
abc
def
ghi
```

- Reading in a batch

```
>>> file=open('~/Desktop/abc.txt')
>>> type(file)
<class '_io.TextIOWrapper'>
>>> s=file.read()
>>> s
'abc\ndef\nghi'
>>> file.close()
>>> file=open('~/Desktop/abc.txt')
>>> l=file.readlines()
>>> l
['abc\n', 'def\n', 'ghi']
>>> file.close()
```

WestlakeNLP

Manipulating Files

- Text files

~/Desktop/abc.txt

```
abc
def
ghi
```

- Reading incrementally & save memory

```
>>> file=open('~/Desktop/abc.txt')
>>> s=file.readline()
>>> while s:
...     print(s)
...     s=file.readline()
...
abc

def

ghi
>>> file.close()
```

WestlakeNLP

Manipulating Files

- Text files

~/Desktop/abc.txt

```
abc
def
ghi
```

- Reading incrementally

```
>>> file=open('~/Desktop/abc.txt')
>>> for line in file:
...     print(line)
...
abc

def

ghi
>>> file.close()
```

WestlakeNLP

Manipulating Files

- Text files

~/Desktop/abc.txt

```
abc
def
ghi
```

- Appending content

```
>>> file=open('abc.txt','a')
>>> file.write('jkl')
3
>>> file.write('\n')
1
>>> file.write('mno\n')
4
>>> file.write('pqr')
3
>>> file.close()
```

WestlakeNLP

Manipulating Files

- Text files

~/Desktop/abc.txt

```
abc
def
ghi
```

- Appending content

```
>>> file=open('abc.txt','a')
>>> file.write('jkl')
3
>>> file.write('\n')
1
>>> file.write('mno\n')
4
>>> file.write('pqr')
3
>>> file.close()
```

- Result

~/Desktop/abc.txt

```
abc
def
ghi
jkl
mno
pqr
```

Manipulating Files

- Text files

~/Desktop/abc.txt

```
abc
def
ghi
```

- Rewriting content

```
>>> file=open('~/Desktop/abc.txt','w')
>>> file.write('1234567')
7
>>> file.close()
```

WestlakeNLP

Manipulating Files

- Text files

~/Desktop/abc.txt

```
abc
def
ghi
```

- Rewriting content

```
>>> file=open('~/Desktop/abc.txt','w')
>>> file.write('1234567')
7
>>> file.close()
```

- Result

~/Desktop/abc.txt

```
1234567
```

WestlakeNLP

Manipulating Files

- Files and Data Structures

~/Desktop/scores.txt

```
1000101 95 90 70 90 85 65
1000103 99 90 77 85 97 55
1000208 70 35 52 56 60 51
```

```
>>> s1="123 456 789"
>>> s1.split()
['123', '456', '789']
>>> s2="abc$def$ghi"
>>> s2.split('$')
['abc', 'def', 'ghi']
```

```
>>> def load_scores(path):
...     d={}
...     file=open(path)
...     for line in file:  # one student item
...         line=line.split()  # split into list
...         d[line[0]]=line[1:]
...     file.close()
...     return d
...
>>> scores=load_scores('~/Desktop/scores.txt')
>>> scores
{'1000101': ['95', '90', '70', '90', '85', '65'],
'1000103': ['99', '90', '77', '85', '97', '55'],
'1000208': ['70', '35', '52', '56', '60', '51']}
```

Manipulating Files

- Files and Data Structures

```
>>> def save_scores(d,path):
...     file=open(path,'w')
...     for k in d:
...         file.write(k)
...         file.write(' ')
...         file.write(' '.join(d[k]))
...         file.write('\n')
...     file.close()
...
>>> d={'1000101':['95','90','70','90','85','45','
    70'],'1000103':['99','90','77','85','97','55'
    ,'71'],'1000208':['70','35','52','56','60','
    51','40']}
>>> save_scores(d,'~/Desktop/scores.txt')
```

~/Desktop/scores.txt

```
1000101 95 90 70 90 85 65
1000103 99 90 77 85 97 55
1000208 70 35 52 56 60 51
```

Manipulating Files

- Files and Data Structures

```
>>> d={'1000101':['95','90','70','90','85','65'],
    '1000103':['99','90','77','85','97','55'],'
    1000208':['70','35','52','56','60','51']}
>>> import pickle
>>> file=open('~/Desktop/scores.txt','wb')
>>> pickle.dump(d,file)
>>> file.close()
>>> file=open('~/Desktop/scores.txt','rb')
>>> d=pickle.load(file)
>>> d
{'1000101': ['95', '90', '70', '90', '85', '65'],
     '1000103': ['99', '90', '77', '85', '97', '
    55'], '1000208': ['70', '35', '52', '56', '60
    ', '51']}
>>> file.close()
```

~/Desktop/scores.txt

```
This file is binary or uses
an unsupported text
encoding, so it cannot be
displayed in a text editor.
```

This week check-off:

Class and Object

WestlakeNLP