Main memory : store data, fast and temporary storage Secondary memory: slower but large size, permanent storage.

CPU:execute your program. (cu+ALU)

CU is used to fetch commands from the memory. ALU contains the electric circuits which can execute command NRZL: 由0跃迁；NRZI：由1越阡。

Assembly language is a low-level programing language.(one to one, Convert by assembler)

High level language: C(highest efficiency), C++, Java(platform independent). High level can't executed directly, change to low. Low with higher language efficiency (run fast), high level —-high development efficiency.

进制转换：1:其他转10: 小数点，左边0起右边-1减。2:10 — 2:

8—2: 0-000, 1-001, 2-010,3-011,4-100, 5-101,6-110,7-111.

16—2: 1-0001, 2-0010, 3-0011,4-0100,5-0101,6-0110,7-0111,8-1000, 9-1001,A-1010,B-1011,C-1100,D-1101,E-1110,F-1111

TB-GB-MB-KB-B,大到小, *1024

Little-endian (increasing):lsb->msb

Big-endian: msb->lsb

Interpreter: program execute. Compiler: convert.

'false' not reserved word.

Variable is a name space in memory to store data, value can change but ID can't. Only '_/l/8' can in name, case sensitive. Different variables may have same name with different scope.

divmod()->(整数,余)

int('10.1') — error, int('10')—10. Only numbers in str.

Boolean type, 0-False, others -True

Is, isnot , same as ==, but check memory address, list always different.

Argument without default value should be front of those have. eg: (x, y=1)

For Strings: index is integer or expression, The second index is up to not reach, second can exceed length.

Build-in function, does not modify the original string, but return a new one.

Eg:s=s.lower();upper()

find(,idx):search for sub string, return the first occurrence index. If not find, then give -1.

replace(): search and replace, will replace all. eg: 'hell'.replace('l','a')—>'heaa'. (do not modify the previous variable)

Stripping whitespace: lstrip()—left, rstrip()—right. strip()—both.

Startswith(): start with a letter or substring. Return True or False.

Open files:

handle=open(name,'r')

Mode 'r'-read,'w'-write

Newline: \n(string), only one character.

Handle file is a sequence of strings, each line is a string. Can also read the whole file into a single string. : all=fhand.read()

When doing writing, fhand.wirte('aa\n')

List: A list can contain any type of data, also empty. Access to data by index. Lists are mutable, (a[0]=1), but strings are not mutable.

+ same as append.

l.index()—give index

l.insert(idx,e); l.pop(idx)

l.sort() —- sort yourself

Sort, sorted have reverse

sorted(l) —- return a new

split(): break a string to a list. Eg: s='a,b,c'

a=s.split()—> a=['a,b,c']

a=s.split(',')—>a=[a,b,c]

中间用于分离的会被省略

range() returns a list of numbers, eg: range(2,4)-2,3; range(0,3,2):0,2.

Dictionary:

Different name: associative arrays(perl) Properties/Maps/hashmap(Java) property bad(c++)

- No index, no order in dictionary, use key to search and change. But key cannot be list type.
- Use key to check whether is in dictionary
- To do counting:

```
d={}
If word in d:
    d[word]=d[word]+1
Else:
    d[word] = 1
```

- get() method:

Dic.get('a',0) —default

若有则出，无则出0

- Loop in dict: for key in dict; for key, value in dict.items():

print(list(dict)) —keys

print(list(dict.keys())

print(list(dict.values())

print(list(dict.items())—tuples

Tuples: immutable

At least have 2 characters, use',' to take position : a=(1,) — 1 element. But a=(1) —int.

- No sort, append, reverse.
- Simpler and efficient in memory use than list.
- Comparable. Begin from the first element, and can use sorted(list) if want to sort by values, then (v,k)..
- Find the 10 most common:

```
fhand=open('txt','r')
ct=dict()
For line in fhand:
  words=line.split()
  For w in words:
     ct[w] =ct.get(w,0)+1

For k,v in ct.items():
     L.append((v,k))
L.sort(reverse=True)
For v,k in l[:10]:
     print(k,v)
```

Object: everything is an object, have unique ID. ID cannot change during the executed, type can change.

- Variable is only a reference to object.
- Methods can only be invoked by specific object.
- Identity: unique id.
- state: properties/ attributes, represent by variables, call data field.

Class: (contract)

use variables to store data fields and defines methods to perform actions.

Object is an instance of class, have lots instances. 同一class的不同instances可有不同的data fields.

Create an instance is instantiation. Objects are interchangeably.

__init__(): initializer. To initialize a new object's state when created.

__new__(): Constructor: 1.create an object in memory; 2.invoke__init__().

Self: all methods have self parameter, refer to the object invokes method.

Data fields also called instance variables, each object has a specific value for a data field. Access to data fields by object member access operator.eg: A.getArea()

Scope:

Instance variable — entire class. (self..)

Local var only within method.

Private data field:

Begin with two'__'. Not end with.. Only can be access within class!! But we can define a method to access.

Abstraction: separate the implementation of code from usage. Details are invisible for user, are encapsulated.

Superclass : inheritance enables to define a general class. — extend to specific class (subclasss).

Different class may have same properties and behaviors.

Subclass inherit accessible methods and data fields from super, also has its own methods. Not a subclass of super!!

Inheritance models 'is-a' relationship.

继承语句：eg:

Circle(geometric)

Super().__init__()—-承 data field.

Overriding: modify a method from super.

Eg: def __str__(self):

Return super().__str__()+'r:' + str(r)

(without super will call itself's __str__)

Every class is a subclass of object, by default.

__new__() in object has the step to call init(). If you override new, then init will not be called.

Polymorphism: an object of a subclass can be passed to a parameter of a super type. Every instance of a subclass is also an instance of super.

Dynamic binding:

A method can be implemented along inheritance chain in several classes. eg：c为最小的sub，找一个method f，先c—c1—c2—…-cn, 直到找到为止。

Isinstance(obj,clasName) : determine whether an object is an instance of class.

Multiple Inheritance:

Define a class from multiple classes.

Eg: C(A,B)

A.__init__(self,a)

B.__init__(self,b)

而不能直接super().__

Data Structure:a systematic way of organizing and accessing data.

Algorithm: a step by step procedure for performing some tasks in a finite time.

A good algorithm includes running time and space usage.

Both time and usage depend on the size of input, input itself, and the hardware, software.

Principle of analysis:

1:Counting primitive operations.

(Assigning an identifier to an object

Determining the object associated with an identifier

Performing an arithmetic operation (for example, adding two numbers)

Comparing two numbers

Accessing a single element of a Python list by index

Calling a function (excluding operations executed within the function)

Returning from a function.)

2.measure as f(n)

3.focus on the worst input.

7 functions for comlexity: c<logn<n<nlogn<n^2<n^3<..<2^x<e^x.

Asymptotic Analysis: Oh notation. If f(n)<cg(n).所有的和n无关的常数及系数均可忽略不计。 Eg:8n—n

Efficient —polynomial complexity. Inefficient —exponential time. Distinction between these is a robust measure of tractability.

Sorting will be showed at the end!!

Recursion: functions makes one or more calls to itself. Contains one or more base cases (non-recursive), also >=1 recursive cases.

Implement recursion: activation record/frame created to store information about the progress, parameters and local variables. If a nested call, the execution of former will suspend until we get the value.时间复杂度为logn说明使用的为二分法!

Eg: 1. Factorial : (n=0)(n(n-1)!). combination

2. Product (), Power()

Binary search: O(logn) (when the sequence is sorted) low and high, and mid; each time compare with mid, less—-low,mid-1; larger—-mid+1,high. same—return.

Linear recursion: at most one call to itself.

Binary sum:

Stack: LIFO(都动顶层) can insert any time, but only access or remove the most recently one.

Queue: FIFO, can insert anytime at top, but only access to the longest time one.

Referential structure: python has a space to store reference, and another store data.

b[0]=a[1] —>id same.

When change a list, the original existing id not change, only change idx

compact structure's memory usage is low.

Linked list: collections of nodes, node store the element and the reference to next node.

Tree: Store data hierarchically.(set of nodes). The top element, only has children without parent, is called root. Other all have parents, children can have one, none, or more.

Edge: a pair of nodes (u,v) such that u is the parent of v (or vice.)

Path: a sequence of nodes such that any two consecutive nodes in the sequence form an edge.

Depth is between root and v.

Leaf node: has no child

Internal node:>=1child

Binary Tree: <=2child, left and right child, left is precede the right.
Tree is proper when it only has 0/2 child.
Tree bases on nodes, each with 3 parameters(parent, lchild, rchild)

DFS: (depth first search)

BFS: sort as bread as you can. Go through all nodes in the same level then go to the next level.

Bubble Sort: iterate over the list, compare every element I with i+1, if I is larger, then swap. Iterate over again and again, each time ignore the last element, until only one element left.

Quick sort: pick a pivot I, i<j—>j-=1,

Notice that there are only one node in tree do not have parents should given tree is none empty!

Sample code:

1. Using dictionary do counting:

```
Tmp={}
For k,v in d.items():
    Tmp.append((v,k))
counts={}
For word in words:
    Counts[word]=
    counts.get(word,0)
    +1
```

Class;

```
2.Class A:
      Def _init_(self,i=1):
           Self.i=i
     Def m1(self):
          Self.i+=1
Class B(A):
     Def _init_(self,j=0):
          super()._init_(3)
          Self.j=j
    Def m1(self):
         self.i+=1
b=B()
B.mi()
print(b.i)
print(b.j)
```

```
4,0
```

```
2. Class A:
     Def __new__(self):
        self.__init__(self)
        print('A new')
     Def __init__(self):
         print('A init')
Class B(A):
    Def __new__(self):
       self.__init__(self)
       print('B new')
     Def __init__(self):
        print('B init')
b=B()
a=A()
```

```
B new,B init, A new, A init.
```

若没有黄线行，则输出仅为 B new,   A new

```
3. Class C1:
      Def _init_(self):
          Self.f=1
     Def output(self):
         print('in c1', self.f)
Class c2(c1):
    Def _init_(self):
        Self.f=2
    Def output(self):
        print('in c2',self.f)
Class c3(c2):
    Def _init_(self):
        Self.f=3
Class c4(c3):
    Def _init_(self):
        Self.f=4
a=c4()
print(a.f)
a.output()
```

```
4, in c2, f is 4.
```

```
3. Class student:
    Def __str__(self):
        Return 'student'
   Def printStudent(self):
       print(self.__str__())
Class graduate(student):
    Def __str__(self):
        Return 'graduate '
a=student()
b=graduate()
a.printStudent(0
B.printStudent()
```

```
Student, graduate
```

若两个黄线处改为__str(), 则输出的为两个student.

```
4. Class person:
      Def getInfo(self):
          Return 'person'
     Def printPerson(self):
         Print(self.getInfo())
Class student(person):
     Def getInfo(self):
         Return 'student'
person().printPerson()
student().printPerson()
```

```
Person, student
```

若黄色处改为 __getInfo(self), 则会输出两个person

```
5. Class A():
   Def _init_(self.a=100):
       Self.a=a
Class B():
  Def _init_(self,b=200):
      self.b=b
Class C(A,B):
   Def
_init_(self,a,b,c=300);
      A.__init__(self,a)
      B.__init__(self,b)
      Self.c=c
  Def output(self):
      print(self.a)
      print(self.c,self.b)
c=C(1,2,3)
c.output()
```

```
1,3,2
```

```
6. Factorial:
 Def facF(n):
   If n<0:
     print('Invalid')
   Elif n==0:
       Return 1
   Else:
       Return n*facF(n-1)
```

```
7.power:
Def power(x,n):
  If n==0: Return 1
   Else:
       p=power(x,n//2)**2
       If n%2==1: p=p*x
```

```
8. DFSearch:
Def Dfsearch(t):
   If t:
       print(t.element)
    If t.left is None and
t.right is None:
         Return
    Else:
       If t.left is not None:
          Dfsearch(t.left)
       If t.right != None:
          Dfsearch(t.right)
```

```
9.BFSearch:
Def BFSearch(t):
    Q = listQueue()
    Q.enqueue(t)
    While Q.size>0:
       cNode=q.dequeue()
      If cNode.left!=None:
  Q.enqueue(cNode.left)
    If cNode.right!=None:
  Q.enqueue(cNode.righ)
    print(cNode.element)
```