Introduction to Computer Science: Programming Methodology

CSC1001 — Python 许烨辰

Slide 1

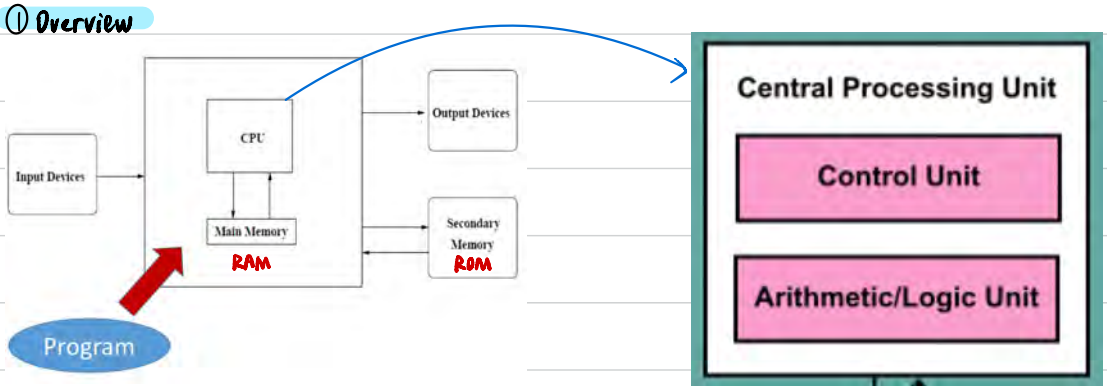
1. Introduction

① Why programming?

Programming languages ⇒ Machine language ⇒ Make the electronic computer know our meanings

2. Hardware

① Overview



② Details

a. Input devices : Mouse, keyboard, panel, touch screen…

b. Output devices : Screen, audio output…

c. Central processing unit (CPU)

△ Description

i) Its function is executing the program (similar to the brain)

ii) Fast but stupid

△ Components

i) Control unit (CU) : Fetch commands from the memory

ii) Arithmetic / logic unit (ALU) : Execute commands

a. Main memory : Fast but small, temporary storage

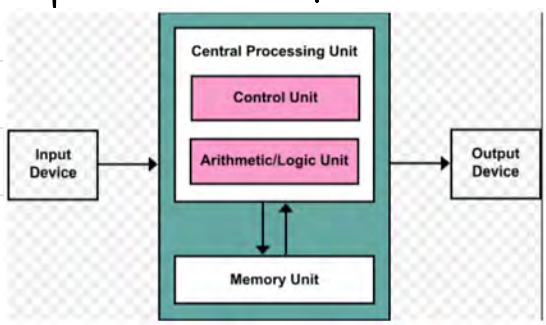
e. Secondary memory : Slow but large, permanent storage

③ Two persons

a. Von Neuman 冯诺伊曼

Proposed the modern computer architecture

Hardware



b. Alan Turing 图灵

- Built theoretical foundation of computer science
- Built artificial intelligence
- Built computability theory and Turing test

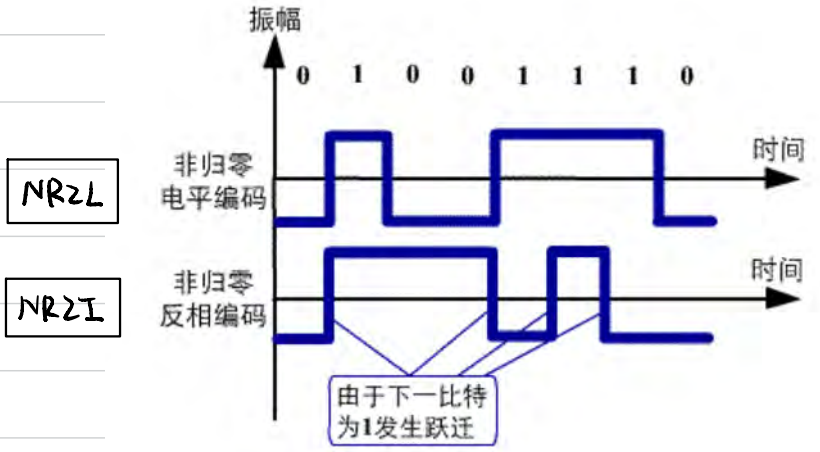
Software

3. Softwares

① Principle

The computer can only understand binary number (0 and 1)

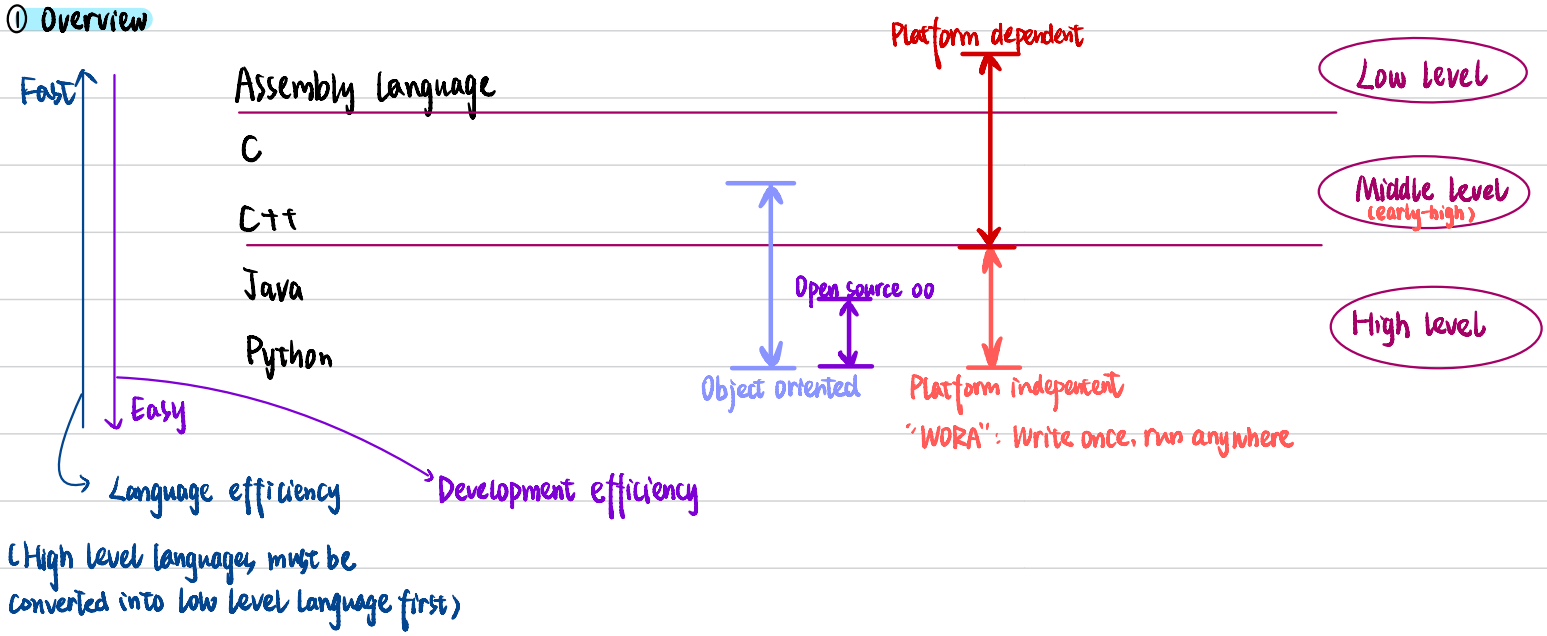
② Two coding methods



③ The instructions expressed in binary code is called machine language (lowest level language)

4. Languages

① Overview



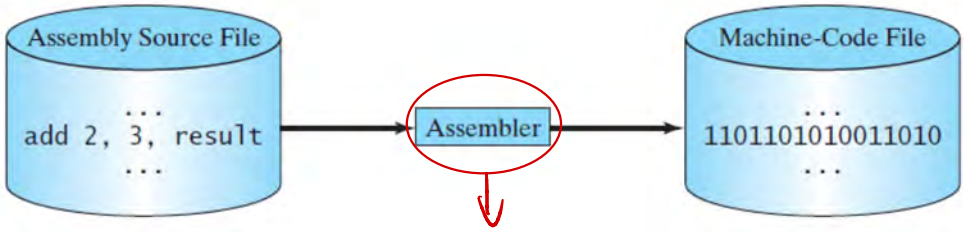
(High level languages must be converted into low level language first)

② Other details

a. Assembly language

⚠ Directly correspond to the machine language

⚠ About "Assembler"



Should be written in machine language

b. C

⚠ Developed at Bell Labs

c. C++

⚠ Developed at Bell Labs

⚠ Inherent major features of C

d. Java

e. Python

⚠ Open source object oriented

⚠ Powerful libraries

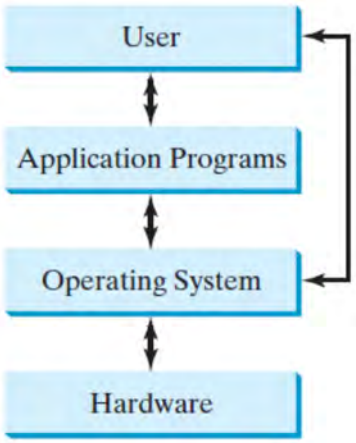
Collection of the codes

⚠ Integrate programming languages (C / C++ / Java / Others)

5. Operating systems

① Definition

The operating system (OS) is a low level program, which provides all basic services for managing and controlling a computer's activities



② Examples

Windows, Mac OS, Linux, iOS, Android

b. Data representation

① Definition

We use positional notation (进位记数法) to represent data.

② Usual number system

Binary (二)、octal (八)、decimal (十)、hexadecimal (十六)

Binary number is the easiest one, it has the lowest cost, So we choose binary number.

③ Basic ideas of positional notation

a. Two elements: Base; Set of symbols $0, 1, \cdots, 9, A, \cdots F$

b. Every number can be decomposed into a weighted sum

- The multiplier is position value
- The weight is $(\text{base})^n$

$$N = a_n \times b^n + \cdots + a_0 \times b^0 + a_{-1} \times b^{-1} + \cdots$$

(a_0: 小数点前; a_{-1}: 小数点后)

c. Conversion

⚠ Method: By definition, $N =$ Weighted sum 1 / Weighted sum 2

⚠ Examples

i) $(24.6)_8 = (2\times8^1 + 4\times8^0 + 6\times8^{-1} + 7\times8^{-2}) = (20.859375)_{10}$

ii) $(0.875)_{10} = (1\times2^{-1} + 1\times2^{-2} + 1\times2^{-3}) = (0.111)_2$

iii) $(1101101.011)_2 = (1\times2^6 + 1\times2^5 + 1\times2^3 + 1\times2^2 + 1\times2^0 + 1\times2^{-2} + 1\times2^{-3})$

$= 64 + 32 + 8 + 4 + 1 + \frac{1}{4} + \frac{1}{8} = 109.375$

$= (1\times8^2 + 5\times8^1 + 5\times8^0 + 3\times8^{-1}) = (155.3)_8$

iv) $(11101.01)_2 = (1\times2^4 + 1\times2^3 + 1\times2^2 + 1\times2^0 + 1\times2^{-2})$

$= 16 + 8 + 4 + 1 + \frac{1}{4} = 29.25$

$= (1\times16^1 + 13\times16^0 + 4\times16^{-1}) = (1D.4)_{16}$

④ Units of data

Bit : A binary digit which takes either 0 or 1 (the smallest data unit in programming)

Byte : 1 Byte = 8 Bits, every English character is represented by 1 byte

- KB : $1\ KB = 2^{10}\ B = 1024\ B$
- MB : $1\ MB = 2^{20}\ B = 1024\ KB$ (以2^{10}为进制)
- GB : $1\ GB = 2^{30}\ B = 1024\ MB$
- TB : $1\ TB = 2^{40}\ B = 1024\ GB$

⑤ Memory and addressing

a. A computer's memory consists of an ordered sequence of bytes for storing data

(A block consists of 1 byte / 8 bits) 每个block对应1个byte

b. The address is unique

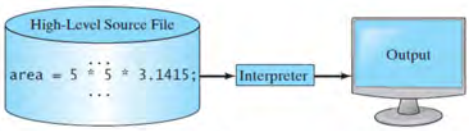
c. The difference between high and low level programming language is whether programmer has to deal with memory addressing directly.

Memory address (Numbering)	Memory content	
.	.	
.	.	
.	.	
2000	01000011	Encoding for character 'C'
2001	01110010	Encoding for character 'r'
2002	01100101	Encoding for character 'e'
2003	01110111	Encoding for character 'w'
2004	00000011	Encoding for number 3
.	.	

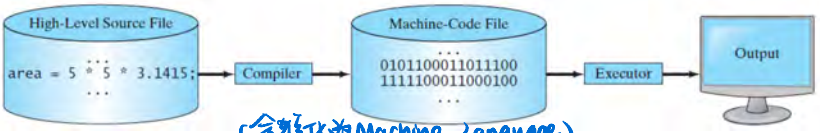
Lecture 2

1. Python basics

① Interpreter vs Compiler



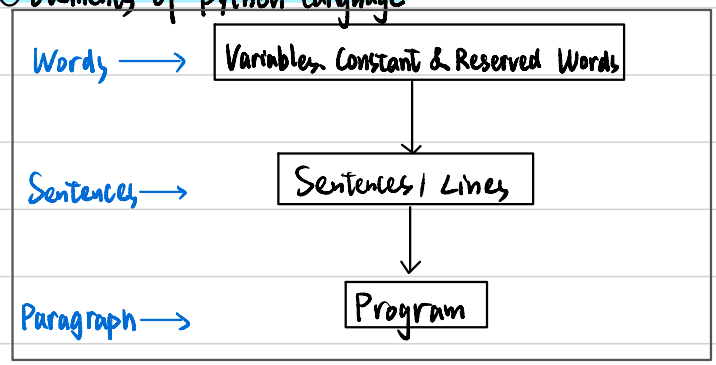
(a)



(會對應成為 Machine Language)

- Interpreter : Line by line ⟶ Good for being platform independent ⟶ Python
- Compiler : The whole program ⟶ Faster ⟶ C / C++

② Elements of python language



a. Constants

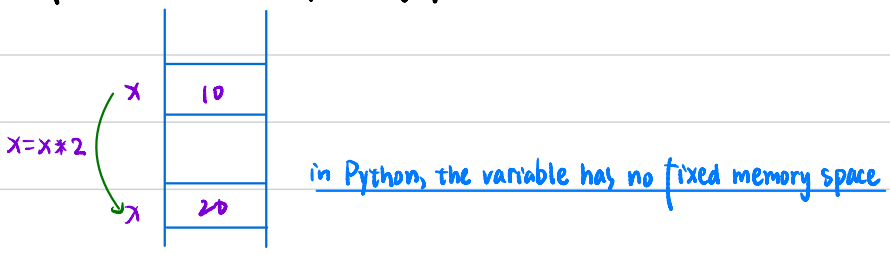
△ Types

Number (Integer, Floating-point number) , letters, string

b. Variables

reference

△ Definition: the name of memory space



in Python, the variable has no fixed memory space

△ Rules of defining variables

i) Start with a letter or "_"

ii) Contain only letters, numbers and "_"

iii) Case sensitive

c. Reserved words / Key words

```
False      None     True     and      as       assert   break
class      continue def      del      elif     else     except
finally    for      from     global   if       import   in
is         lambda   nonlocal not      or       pass     raise
return     try      while    with     yield
```

d. Sentences → Assignment statement

⚠ Structure

```
x = x * 2
```

variable — assignment operator — expression

⚠ Other types

i> Cascaded assignment

Example

```
>>> z = y = x = 2 + 7 + 2
>>> x, y, z
(11, 11, 11)
```

ii> Simultaneous assignment

Example

```
>>> c = "deepSecret"           # Set current password.
>>> o = "you'll never guess"   # Set old password.
>>> c, o                       # See what passwords are.
('deepSecret', "you'll never guess")
>>> c, o = o, c                # Exchange the passwords.
```

同时把对值，无需引入第三变量

iii> Augmented assignment

Example

```
>>> x = 22        # Initialize x to 22.
>>> x += 7        # Equivalent to: x = x + 7
>>> x
29
>>> x -= 2 * 7    # Equivalent to: x = x - (2 * 7)
>>> x
15
```

x = x + 7

x = x - 14

③ Program flow

a. Sequential flow

```
>>> x=2
>>> print(x)
2
>>> x=x*10
>>> print(x)
20
>>>
```

Outputs

i> 把代码里的结构看下一个整体 (Block / Body)

ii> if → when it is true → executed once

while → repeats

b. Conditional flow

Program

```
x=5
if x<10:
    print("smaller")
if x>20:
    print("bigger")
print("finished")
```

Outputs

```
smaller
finished
>>>
```

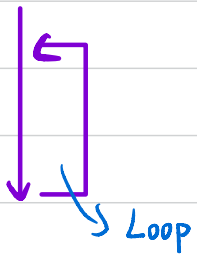
c. Repeated flow

Program

```
n=5
while n>0:
    print(n)
    n = n - 1
print("Finish")
```

Outputs

```
5
4
3
2
1
Finish
>>>
```



④ Arithmetic operation

a. For numbers

△ ** 指数

△ % 求余

△ // 去求商 (floor division)

```
>>> 9 // 2.5
3.0
```

△ divmod() 求商以求余

```
>>> time = 257         # Initialize time.
>>> SEC_PER_MIN = 60 # Use a "named constant" for 60.
>>> divmod(time, SEC_PER_MIN)   # See what divmod() returns.
(4, 17)
```

Example:

```
>>> a = 100 + 200
>>> print(a)

>>> b = "100" + "200"
>>> print(b)
```

```
#Data type(p48)
a=100
b=200
c="300"
d="400"
#注意一下区别:
#print(a+b)->300
#print(a,b)->100_200
#print(c+d)->300400
#print(c,d)->300_400
```

b. For strings

△ + : Concatenation

△ * : Repetition

△ [ ] : Slice: Shows the character in the string

[:]: Range slice

```
>>> a='test1'
>>> b='test2'
>>> a+b
'test1test2'
>>> a*3
'test1test1test1'
>>> a[0]   从0开始
't'
>>> a[0:3]   Slice 若与前为str若体
'tes'
>>> a*b
Traceback (most recent call last):
  File "<pyshell#168>", line 1, in <module>
    a*b   str不能乘str
TypeError: can't multiply sequence by non-int of type 'str'
>>> a**3
Traceback (most recent call last):
  File "<pyshell#169>", line 1, in <module>
    a**3
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int'
```

⑤ Data type

a. Definition

Constants and variables have an associated "type", it is called "Data type" (class)

b. Types

- Integer
- Floating-point number
- Complex number
- String
- Boolean (不用记) : the value can only be "True" or "False" (upper case) (comparison assignment in)
- List e.g. [1,2,3], ['h','e','l','l','o'] (inside is number / string)

Data types can be distinguished : type ( )

Data types can be changed: (9.8->9 不用记10)

e.g. int(9.8)->9, str(123)->'123', float('9.8')->9.8,
list('hello')-> ['h','e','l','l','o'] ...

int( ), float( ), str( ), list( )

number --int( )--> integer

number / string (only number in string) --float( )--> floating-point number

number --str( )--> string

string --list( )--> list

⑥ Basic functions

a. Print( )

⚠ Print(value, sep, end)

```
>>> a=1
>>> b=2
>>> c=3
>>> print(a, b, c)
1 2 3
>>> print(a, b, c, sep=':')
1:2:3
>>> print(a, b, c, end='.')
1 2 3.
```

◬ Formatted output

i) Three types (change the output type)

- Integer : print("%d"%v)
- Floating-point number : print("%f"%v)
- String : print("%s"%v)

variable name

```
>>> a=1.0
>>> print('%d'%a)
1
>>> print('%f'%a)
1.000000
>>> print('%s'%a)
1.0
```

ii) Spaces and digits after decimal points

the number('8') before 'd' in print('%8d'%v) means 8 spaces for the integer v;
the number('7.2') before 'f' in print('%7.2f'%v) means 7 spaces for the floating point number v while keeping the 2 digits after decimal point '.'.

To align the result leftward, add the symbol '-': print('%-8d'%v) will make the integer v printed from the left of the 8 spaces. Without '-' will align the result rightward.

b. input( )

input( ) is a function used for instructing the users to input a string for a variable

input( ): str → variable (str data type)

```
>>> a=input('Please enter a number: ')
Please enter a number: 2
>>> a
'2'
>>> type(a)
<class 'str'>
```

If we want the other types, we need to convert it.

c. eval( )

eval( ) is a function used to evaluate the expression of string eval("exp") = Result of the expression

eval( ) is often used with input( )

eval(input( )): str → variable (exclude str type) (as if remove the " ")

```
>>> a=input('Please enter a number: ')
Please enter a number: 1
>>> type(a)
<class 'str'>
>>> b=eval(input('Please enter a number: '))
Please enter a number: 1
>>> type(b)
<class 'int'>
```

字符串 expression

```
>>> input('Please enter a number:')
Please enter a number:test
'test'
>>> eval(input('Please enter a number: '))
Please enter a number: test
Traceback (most recent call last):
  File "<pyshell#199>", line 1, in <module>
    eval(input('Please enter a number: '))
  File "<string>", line 1, in <module>
NameError: name 'test' is not defined
```

Lecture 3

1. Conditional flow

① Comparison operators

- < / >
- <= / >=
- == (= : Assignment; == : Comparison)
- != ("≠")

```
>>> 5 > 7                  # Is 5 greater than 7?
False
>>> x, y = 45, -3.0
>>> x > y                  # Is 45 greater than -3.0?
True
>>> result = x > y + 50 # Is 45 greater than -3.0 + 50?
>>> result
False
>>> if 1 + 1 > 1:
...     print("I think this should print.")
...
I think this should print.
>>> "hello" > "Bye"        # Comparison of strings.
True
>>> "AAB" > "AAC"
False
```

Lower case > Upper case
Latter letter > Former letter
(先编不号时，先编前面时)

```
>>> 7 == 7.0
True
>>> x = 0.1
>>> 1 == 10 * x
True
>>> 1 == x + x + x + x + x + x + x + x + x + x
False
>>> x + x + x + x + x + x + x + x + x + x
0.9999999999999999
>>> 7 != "7"
True
>>> 'A' == 65
False
```

(乘法 > 加法)

Decimal → Binary → Decimal

a. Priority: Arithmetic operation > Comparison operation;

b. String comparison: Each character has its order, compare them one by one → (if inequal, stop; if equal, compare the next one)

② Boolean bool( )

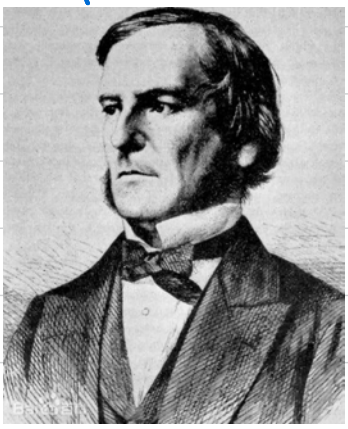
a. Two values:

True / False

```
>>> x = 0; y = 0.0; z = 0 + 0j
>>> bool(x), bool(y), bool(z)
(False, False, False)
```

False: 0 (Regardless of the number type)

True: all other numbers



③ Logical operators

a. Types

not (¬), and (∧), or (∨)

b. Function

Combine several boolean expression into a single boolean expression

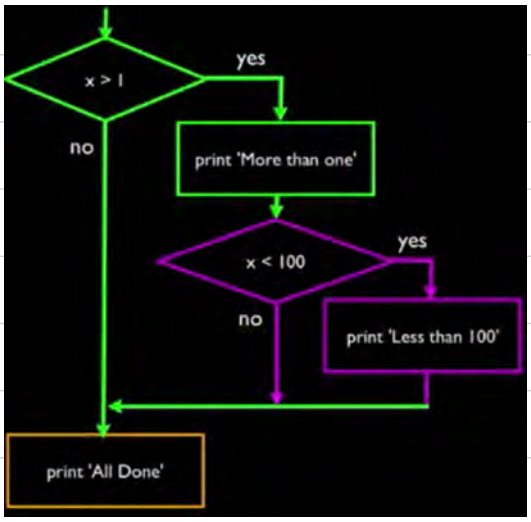
```
>>> not True
False
>>> False and True          T ∧ F
False
>>> not False and True
True
>>> (not False) and True
True
>>> True or False           T ∨ F
True
```

```
>>> not False or True
True
>>> not (False or True)
False
>>> False and False or True
True
>>> False and (False or True)
False
```

①Flowchart

◇ : Check the condition

▭ : Common instruction



⑤ Four types of decisions

a. One-way decision X way decision ⇒ X ways out

⚠ Reserved words: if

```
x=5
print('Before 5')
if x==5:
    print('Is 5')
    print('Is still 5')
    print('Third 5')

print('Afterwards 5')
```

b. Two-way decision

⚠ Reserved words: if、else

```
x=1

if x>2:
    print('Bigger')
else:
    print('Smaller')

print('Finished')
```

if & else both have their own body

```
#No else

x=2
if x<2:
    print('Small')
elif x<10:
    print('Medium')

print('Finished')
```

c. Multi-way decision

⚠ Reserved words: if、elseif (can be multiple)、else (not compulsory)

```
x=2
if x<2:
    print('Small')
elif x<10:
    print('Medium')
else:
    print('Large')

print('Finished')
```

Execute in order (just one output)

d. Nested decision

```
x=42
if x>1:
    print('More than 1')
    if x<100:
        print('Less then 100')

print('Finished')
```

When we want to verify multiple conditions at the same time, we use nested decision

Using indentation to represent different "nests"

⑥ Try / Except Structure

a. Explanation

If "try" works, then "except" is skipped;

If "try" fails, then "except" is executed.

b. Function

Capture errors

```
astr = 'Hello bob'
istr = int(astr)
print('First', istr)

astr = '123'
istr = int(astr)
print('Second', istr)
```

Fail to run

```
astr = 'Hello bob'
try:
    istr = int(astr)
except:
    istr = -1
print('First', istr)

astr = '123'
try:
    istr = int(astr)
except:
    istr = -1
print('Second', istr)
```

Succeed to run

```
rawstr = input('Enter a number:')

try:
    ival = int(rawstr)
except:
    ival = -1

if ival>0:
    print('Nice work')
else:
    print('Invalid number')
```

2. Repeated flow

①Indefinite loop (execute uncertain number of time)

a. Reserved word: while

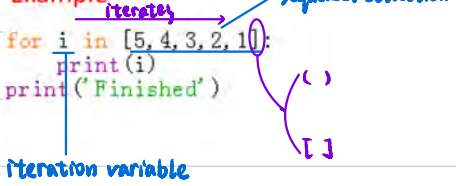
b. Infinite loop

```
n=5
while n>0:
    print('Lather')
    print('Rinse')
n=n-1
print('Dry off!')
```

② Definite loop (execute an exact number of times)

a. Reserved word: for

Example



Output

```
5
4
3
2
1
Finished
```

Example

```
friends = ['Tom','Jerry','Bat']
for friend in friends:
    print('Happy new year',friend)
print('Done')
```

Output

```
Happy new year Tom
Happy new year Jerry
Happy new year Bat
Done
```

```
##for loop
for i in loop-list:
    # Loop body
```

we can firstly define the list (one of the data types)

```
sumup=0
for num in range(1,101):
    sumup+=num
print('the sum from 1 to 100 is', sumup)
```

b. Application

⚠ Process

i) Set some initial storing variable

ii) Loop

iii) Print the output

Examples

i) Finding the largest number

Example

```
largest_so_far = -1
print('Before', largest_so_far)

for num in [9, 39, 21, 98, 4, 5, 100, 65]:
    if num>largest_so_far:
        largest_so_far = num
    print(largest_so_far, num)

print('After', largest_so_far)
```

显示这个循环过程

Output

```
Before -1
9 9
39 39
39 21
98 98
98 4
98 5
100 100
100 65
After 100
```

two variable, Initial storing variable, Iteration variable

ii) Finding the smallest number

```
smallest_so_far = -1
print('Before', smallest_so_far)

for num in [9, 39, 21, 98, 4, 5, 100, 65]:
    if num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far, num)

print('After', smallest_so_far)
```

this loop can not function

Example

```
smallest_so_far = None
print('Before', smallest_so_far)

for num in [9, 39, 21, 98, 4, 5, 100, 65]:
    if smallest_so_far == None:
        smallest_so_far = num
    elif num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far, num)

print('After', smallest_so_far)
```

type(None) → NoneType

Output

```
Before None
9 9
9 39
9 21
9 98
4 4
4 5
4 100
4 65
After 4
```

iii) Counting the # of elements in the collection

Example

```
count = 0
print('Before', count)
for thing in [3, 4, 98, 38, 9, 10, 199, 78]:
    count = count + 1
    print(count, thing)
print('After', count)
```

Output

```
Before 0
1 3
2 4
3 98
4 38
5 9
6 10
7 199
8 78
After 8
```

iv) Filtering the elements in the collection

Example

```
print('Before')

for value in [23, 3, 43, 39, 80, 111, 99, 3, 65]:
    if value>50:
        print('Large value:', value)

print('After')
```

Output

```
Before
Large value: 80
Large value: 111
Large value: 99
Large value: 65
After
```

v) Searching using a Boolean variable

Example

```
found = False

print('Before', found)

for value in [9, 41, 12, 3, 74, 15]:
    if value == 74:
        found = True
    print(found, value)
print('After', found)
```

Output

```
Before False
False 9
False 41
False 12
False 3
True 74
True 15
After True
```

③ break and continue

a. break

"break" ends the current loop, and then jumps to the next statement

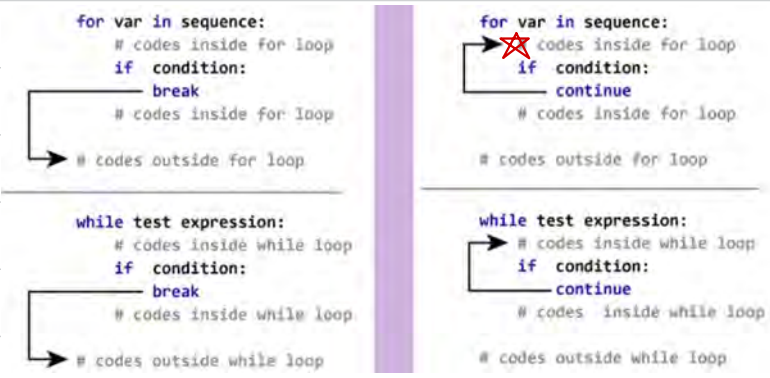
```
while (True):          (artificial infinite loop)
    line = input('Enter a word:')
    if line == 'done':
        break
    print(line)
print('Finished')
```

b. continue

"continue" ends the current loop, and then starts the next iteration

```
while True:
    line = input('Input a word:')
    if line[0] == '#': continue
    if line == 'done':
        break
    print(line)
print('Done')
```

(pay attention to the practical flowchart)



④ is and is not

"is" ≈ "=="

"is not" ≈ "!="

```
print(10 is 10)          True

a = 10
b = 10
print (a is b)           True

a = '123'
b = '123'
print (a is b)           False

a = [1, 2, 3]
b = [1, 2, 3]
print (a is b)
```

collection is an exception

除了string之外的collection

Lecture 4

1. Function

① Definition

The reusable paragraph is called function when we want to reuse a paragraph, we define it

② Types

- Built-in function e.g. print(), int(), float() we can use them as variable name, but they will lose function's property
- User-defined function

③ Structure

a. Overview

```
def Hello():
    print('Hello')
    print('Funny')

Hello()
print('Something in the middle.')
Hello()
```

(Define the function) ← ind var, func, dep var

Call/Invoke the function ←

b. Define the function

⚠1 Argument: A value as input of function (x-value)

```
big = max('I am the one')
```

⚠2 Parameter: A variable as input of function (x)

```
def greet(lang):
    if lang=='es':
        print('Hola')
    elif lang=='fr':
        print('Bonjour')
    else:
        print('Hello')

>>> greet('en')
Hello
>>> greet('es')
Hola
>>> greet('fr')
Bonjour
```

i> # of para/argu

- Single
- Multiple

```
def AddTwo(a, b):
    total = a+b
    return total

x=AddTwo(3, 5)
print(x)
```

ii) Scope of para

- Local variable : Variable in the function
- Global variable : Variable else

global var → local var: use keyword global 调用 Global var

直接改变 Global var

```
x = 1
def increase():
    global x
    x = x + 1
    print(x) # Displays 2

increase()
print(x) # Displays 2
```

```
globalVar = 1
def f1():
    localVar = 2
    print(globalVar)
    print(localVar)
```

iii) Default argument

Set default argument →
- New argument → New argument
- No argument → Default argument

⚠ Return value: The dependent variable of the function (y-value)

```
def greet(lang):
    if lang=='es':
        return 'Hola'
    elif lang=='fr':
        return 'Bonjour'
    else:
        return 'Hello'
```

```
>>> print(greet('en'),'Glenn')
Hello Glenn
>>> print(greet('es'),'Sally')
Hola Sally
>>> print(greet('fr'),'Michael')
Bonjour Michael
```

i) Difference between "return" & "print( )"

- return : y → y-value return will memorize f(x) → 不会输出!!!
- print( ) : → y-value print will not memorize f(x)

```
# Print grade for the score
def printGrade(score):
    if score >= 90.0:
        print('A')
    elif score >= 80.0:
        print('B')
    elif score >= 70.0:
        print('C')
    elif score >= 60.0:
        print('D')
    else:
        print('F')
```

```
>>> def function():
        print("Hello")
        print("World")

>>> n=function()
Hello
World
>>> print(n)
None
>>> def function2():
        print("Hello")
        return
        print("World")

>>> n=function2()
Hello
>>> print(n)
None
```

ii) # of the return values

- Null — Void function
- Single
- Multiple

```
def sort(number1, number2):
    if number1 < number2:
        return number1, number2
    else:
        return number2, number1

n1, n2 = sort(3, 2)
print("n1 is", n1)
print("n2 is", n2)
```

④ Specific functions

a. max( ) & min( )

```
>>> big = max('Hello world')
>>> print(big)
w
>>> small = min('Hello world')
>>> print(small)
 
```

a

str ⟶ max( ) / min( ) ⟶ str

2. String type

① Function

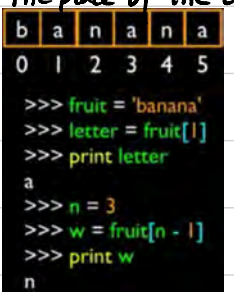
(Parsing

\ Converting

② Index

a. An index

The place of the characters in a string



Starts from 0

Str ⟶ list ⟶ index

If the index is beyond the range, we will get error

```
>>> name = 'Junhua'
>>> name[6]
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    name[6]
IndexError: string index out of range
```

name[-1] ✓

name[-100] ✗

无穷无尽！

b. A range of indices —— Slicing strings

① 左闭右开：上界不取值

② stringname [:]

③ in the range, index can be out of range

④ Blank = ∞

```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
>>> s='Monty Python'
>>> print(s[:6])
Monty
>>> print(s[3:])
ty Python
>>> print(s[:])
Monty Python
```

③ len( )

```
>>> fruit = 'apple'
>>> length = len(fruit)
>>> print(length)
5
```

length = # of the chars in the list = # of indices

④ String methods

a. String library

The collection of string functions = methods

These functions are built-in functions

These functions will not modify the original string, instead they return a new one

b. String methods

① upper( ), lower( )

often used before find( ): no need to worry about case

```
>>> greet = 'Hello, President Xu'
>>> zap = greet.  lower()
>>> print(zap)
hello, president xu
>>> print(greet)
Hello, President Xu
>>> print('Hello, Junhua'.lower())
hello, junhua
```

y = Str.strfunc

② find( )


```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

① Output the first match

Output the first char in the first match

② -1: It means that the sub-string is not found

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2016'
>>> atpos = data.find('@')
>>> print(atpos)
21
>>> sppos = data.find(' ',atpos)
>>> print(sppos)
31
>>> host = data[atpos+1:sppos]
>>> print(host)
uct.ac.za
```

start from atpos 第3个start

replace( )

```
>>> greet = 'Hello, Bob'
>>> newStr = greet.replace('Bob', 'Jane')
>>> print(newStr)
Hello, Jane
>>> newStr = greet.replace('o','X')
>>> print(newStr)
HellX, BXb
>>> newStr = greet.replace('z','X')
>>> newStr
'Hello, Bob'
```

from — to

strip( )

```
>>> greet = '  Hello Bob  '
>>> greet.lstrip()
'Hello Bob  '
>>> greet.rstrip()
'  Hello Bob'
>>> greet.strip()
'Hello Bob'
```

lstrip( )

rstrip( )

strip( )

startswith( )

```
>>> line = 'Please submit your application'
>>> line.startswith('Please')
True
>>> line.startswith('p')
False
```

⑤ Examples

a. Looping through strings

```
fruit = 'I am the one, Morpheus'

n = 0
for i in fruit:
    print(n, i)
    n=n+1

print('finished')
```

take every char in the string

```
0 I
1
2 a
3 m
4
5 t
6 h
7 e
8
9 o
10 n
11 e
12 ,
13 M
14 o
15 r
16 p
17 h
18 e
19 u
20 s
finished
```

space is one kind of character

b. Counting the characters

```
word = 'banana'
count = 0
for letter in word:
    if letter=='a':
        count = count+1
print("The number of 'a' we have seen is:", count)
```

the iteration variable is named after "for"

c. Judging the existence of characters

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit:
        print('Got cha!')
```

不是"is in"

3. File

① Text file

A collection/sequence of strings

② Newline character

"\n" : One character, not two

```
>>> stuff = 'Hello\nWorld'
>>> stuff
'Hello\nWorld'
>>> print(stuff)
Hello
World
>>> stuff = 'X\nY'
>>> print(stuff)
X
Y
>>> len(stuff)
3
```

③ open( ) . close( )

handle = open("filename", "mode")

optional — ("r" "w" "a")

```
>>> fhand = open('c:\Python35\myhost.txt','r')
>>> print(fhand)
<_io.TextIOWrapper name='c:\\Python35\\myhost.txt' mode='r' encoding='cp936'>
```

a. Reading the text file

```
fhand = open('myhost.txt','r')

for line in fhand:
    print(line)

fhand.close()
```

```
fhand = open('myhost.txt','r')
allText = fhand.read()
print('The length of the file:',len(allText))
print('The first 20 characters of the file:', allText[:20])
```

b. Searching through a file

```
fhand = open('myhost.txt','r')

for line in fhand:
    if line.startswith('#')==True:
        print(line)

print('finished.')
fhand.close()
```

c. Writing the text file

```
fhand = open('test.txt','w')
fhand.write('The first line\n')
fhand.write('The second line\n')
fhand.write('The third line\n')
fhand.close()
```

Lecture 5

1. Collection

① Function

- Other data type : Put a value in a variable
- Collection : Put many values in a variable

② Types

- List
- Dictionary
- Tuple

2. List

① Definition

[, , , ···]

the elements can be number、string、another list、blank...

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow', 'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print(1, [5, 6], 7)
1 [5, 6] 7
>>> print([])
[]
```

①和它字个空格，输出种类

②不输[, ,]

③[None] → [None] Nonetype

② Relation with string

a. Similarities

Index

```
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(friends[1])
Glenn
```

Slicing

```
>>> t=[9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41, 12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

len( )

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x=[1, 2, 'joe', 99]
>>> print(len(x))
4
```

Operations

- Concatenation
- Repetition

in / not in

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
```

Differences

- String is not mutable
- List is mutable

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    fruit[0] = 'b'
TypeError: 'str' object does not support item assignment
>>>
>>> x=fruit.lower()
>>> print(x)
banana
>>>
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2]=28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

methods

- String's method will create a new string
- List's method will modify the list itself

because list is mutable

③ range( )

a. Definition

range( ) returns a list of numbers → pseudo list

```
>>> x=range(4)
>>> x
range(0, 4)
>>> x[0]
0
>>> x[1]
1
>>> x[2]
2
>>> x[3]
3

>>> x=range(2, 10, 2)
>>> x[0]
2
>>> x[3]
8
>>> x[4]
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    x[4]
IndexError: range object index out of range
```

① range(4) = range(0,4) = range(0,4,1)

② 左闭右开. 取不到

b. Example

Example

```
friends = ['Tom','Jerry','Bat']

for friend in friends:
    print('Happy new year,', friend)

for i in range(len(friends)):
    friend = friends[i]
    print('Happy new year,', friend)
```

Output

```
Happy new year, Tom
Happy new year, Jerry
Happy new year, Bat
Happy new year, Tom
Happy new year, Jerry
Happy new year, Bat
>>>
```

④ List methods

a. append( )

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

⚠ list( ) is used to create a new blank list

⚠ append in order

b. sort( )

```
>>> friend = ['Tom','Jerry','Bat']
>>> friends.sort()
>>> print(friends)
['Bat', 'Jerry', 'Tom']
>>> print(friends[1])
Jerry
>>>
>>> numbers = [1,2,5,100,32,7,97,1001]
>>> numbers.sort()
>>> print(numbers)
[1, 2, 5, 7, 32, 97, 100, 1001]
```

c. split( )

String —split→ list

```
>>> myStr = 'Catch me if you can'
>>> words = myStr.split()
>>> print(words)
['Catch', 'me', 'if', 'you', 'can']
```

⚠ Multiple spaces are treated like one delimeter

```
>>> line = 'A lot                of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
```

⚠ We can self-define the delimiter

```
>>> thing = line.split(';')
>>> print(thing)
['first', 'second', 'third']
```

3. Dictionary

① Process

a. Create a dictionary : dict( )

b. Set the tag and its value

② Relation with list

```
>>> lst = list()
>>> lst.append(21)
>>> lst.append(183)
>>> print(lst)
[21, 183]
>>> lst[0] = 23
>>> print(lst)
[23, 183]
```

```
>>> ddd = dict()
>>> ddd['age']=21
>>> ddd['course']=182
>>> print(ddd)
{'age': 21, 'course': 182}
>>> ddd['age']=23
>>> print(ddd)
{'age': 23, 'course': 182}
```

```
>>> purse = dict()
>>> purse['money'] = 12
>>> purse['candy'] = 3
>>> purse['tissues'] = 75
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 3}
>>> print(purse['candy'])
3

>>> purse['candy']=purse['candy']+2
>>> print(purse)
{'money': 12, 'tissues': 75, 'candy': 5}

>>> purse[3] = 77
>>> print(purse)
{3: 77, 'money': 12, 'tissues': 75, 'candy': 5}
```

tag: can be all data type

Similarity: Both are mutable

Difference: List is with order, while dictionary is without order

[0,1,2,…]

③ Retrieving lists of keys and values

list(dict.__())

- .keys( )
- .values( )
- .items( )

```
>>> jjj = {'chuck':1,'fred':42,'jan':100}
>>> print(list(jjj))
['jan', 'fred', 'chuck']

>>> print(list(jjj.keys()))
['jan', 'fred', 'chuck']

>>> print(list(jjj.values()))
[100, 42, 1]
>>> print(list(jjj.items()))
[('jan', 100), ('fred', 42), ('chuck', 1)]
```

tuple( ) is also okay

the outside framework is a list, the inside elements are tuples

```
#for & dictionary(p.40)
name_dictionary={"许炜塬":1,"土肥塬":2,"Mickey":3}
for name in name_dictionary:
    print(name)
    print(name,name_dictionary[name])
```

[8] ✓ 0.3s

```
许炜塬
许炜塬 1
土肥塬
土肥塬 2
Mickey
Mickey 3
```

④ Two applications with dictionary

a. Counting

⚠ Original method

```
wordDict = dict()

while True:
    word = input('Enter a word:')

    if word in wordDict:
        wordDict[word]=wordDict[word]+1
    elif word=='done':
        break
    else:
        wordDict[word]=1

print('The result of word count:')
print(wordDict)
```

Use the value to do counting.

⚠ get() method (get() can check whether a string is one of the keys in the dictionary)

```
>>> counts = {'aaa':1,'bbb':2,'ccc':5}
>>> print(counts.get('eee',0))
0
```

→ the output is a number

- the key is existent before → the value won't be changed
- the key is not existent before → the value will be set

Answer

```
wordDict = dict()

textLine = input('Enter a line:')
words = textLine.split()

print('Counting...')
for word in words:
    wordDict[word] = wordDict.get(word,0) + 1

print('The result of word count:')
print(wordDict)
```

dict.items()

b. Sorting

```
>>> d={'a':10,'b':1,'c':22}
>>> tmp = list()
>>> for k,v in d.items():
        tmp.append((v,k))

>>> print(tmp)
[(22, 'c'), (1, 'b'), (10, 'a')]
>>> tmp.sort(reverse=True)
>>> print(tmp)
[(22, 'c'), (10, 'a'), (1, 'b')]
```

Example: Finding the 10 most common words in a file

```
fhand = open('myhost.txt','r')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word,0)+1

lst = list()
for key,val in counts.items():
    lst.append((val,key))

lst.sort(reverse = True)

for val,key in lst[:10]:
    print(key,val)
```

4. Tuple

① Definition

Tuple is the same as list except

- list is mutable
- tuple is immutable (similar to string)

```
>>> y='abc'
>>> y[2]='e'
Traceback (most recent call last)
:
  File "<pyshell#23>", line 1, in
<module>
    y[2]='e'
TypeError: 'str' object does not
support item assignment
```

```
>>> z=(5,4,3)
>>> z[2]
3
>>> z[2]=0
Traceback (most recent call last)
:
  File "<pyshell#28>", line 1, in
<module>
    z[2]=0
TypeError: 'tuple' object does no
t support item assignment
```

⇒ .sort() .append() .reverse()

- list: ✓
- tuple: ✗

```
>>> x=(1,2,3)
>>> x.sort()
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    x.sort()
AttributeError: 'tuple' object has no attribute 'sort'
>>> x.append(5)
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    x.append(5)
AttributeError: 'tuple' object has no attribute 'append'
>>> x.reverse()
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    x.reverse()
AttributeError: 'tuple' object has no attribute 'reverse'
```

The essence of these methods is assignment.

② Why tuple?

- list : mutable but slow
- tuple : immutable but fast

→ tuples are more frequently used

Lecture 6-7

Abstraction (抽象): Separate c

Encapsulation (封装): Data ins
object's methods. Data and meth
parameters to the functions.

Inheritance (继承): Inherit the

Polymorphism (多态): For a fu
which means the references are r

Dynamic Binding (动态绑定):
chain. The method to be invoked
other languages(e.g. Java) may no

Overriding (重载): The same n

1. Object

① Definition

Everything in Python is an object

```
>>> n = 3  # n is an integer
>>> id(n)
505408904
>>> type(n)
<class 'int'>
>>> s = "Welcome" # s is a string
>>> id(s)
36201472
>>> type(s)
<class 'str'>
```

id( ) & type( ) can be used to get information about an object

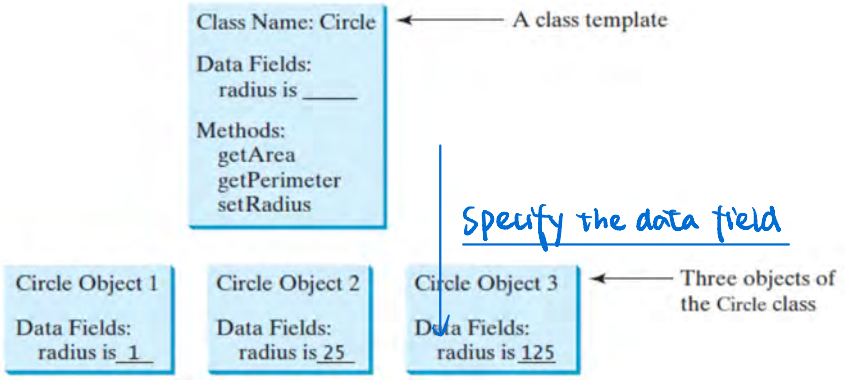
② Three elements

a. Identity

b. State / Data field

c. Behavior / Methods

③ Relation between Class & Objects



2. Class

① Define the class

```
class ClassName:
    initializer
    methods
```

Content (UML)

TV
channel: int volumeLevel: int on: bool
TV() turnOn(): None turnOff(): None getChannel(): int setChannel(channel: int): None getVolume(): int setVolume(volumeLevel: int): None channelUp(): None channelDown(): None volumeUp(): None volumeDown(): None

The current channel (1 to 120) of this TV. The current volume level (1 to 7) of this TV. Indicates whether this TV is on/off.
Constructs a default TV object. Turns on this TV. Turns off this TV. Returns the channel for this TV. Sets a new channel for this TV. Gets the volume level for this TV. Sets a new volume level for this TV. Increases the channel number by 1. Decreases the channel number by 1. Increases the volume level by 1. Decreases the volume level by 1.

```
import math

class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.radius = radius

    def getPerimeter(self):
        return 2 * self.radius * math.pi

    def getArea(self):
        return self.radius * self.radius * math.pi

    def setRadius(self, radius):
        self.radius = radius
```

i) self is always the first parameter of the class, it refers to the object

ii) self.radius is a data field (i.e. parameter)

iii) self.radius → Data field → Outside: object.radius

self.getArea() → Method → Outside: object.getArea()

iv) Scope of the variable: Local (function), Instance (class), Global (program)

② Execute the class

object

when we create an object from a class, we use constructor.

```
>>> circle1 = Circle()
>>> circle1.radius
1
>>> circle1.getPerimeter()
6.283185307179586
>>> circle1.getArea()
3.141592653589793
>>> circle1 = Circle(2)
>>> circle1.radius
2
>>> circle1.radius = 10
>>> circle1.getArea()
314.1592653589793
```

2 functions of constructor:

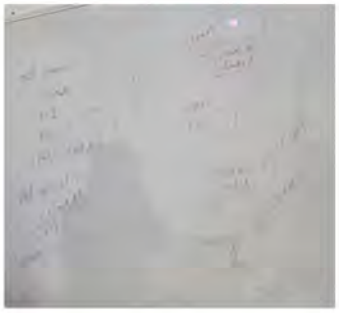
1. It creates an object in the memory for the class.
2. It invokes the class's __init__ method to initialize the object. The self parameter in the __init__ method is automatically set to reference the object that was just created.

object Data Fields:

__init__(self, ...)

self → circle obj1 r=1

Practice



```
class Count:
    def __init__(self, count = 0):
        self.count = count

def main():
    c = Count()
    n = 1
    m(c, n)

    print("count is", c.count)
    print("n is", n)

def m(c, n):
    c = Count(5)
    n = 3

main() # Call the main function
```

0

1

- What would be the output of the above program?

③ Hiding data fields

a. Public & Private DF

- Public : Can be accessed outside the class
- Private : Can't be accessed outside the class , using "__"

```
import math

class Circle:
    # Construct a circle object
    def __init__(self, radius = 1):
        self.__radius = radius

def getRadius(self):
    return self.__radius

def getPerimeter(self):
    return 2 * self.__radius * math.pi

def getArea(self):
    return self.__radius * self.__radius * math.pi
```

④ Class abstraction & encapsulation

- Abstraction : Separating the implementation & the usage of the code
- Encapsulation : Hiding the data fields & methods of the class

封装意味着不允许直接 print 出 Data field，只能通过定义 get XXX 等 method 来获得

```
class A:
    def __init__(self, i):
        self.__i = i

def main():
    a = A(5)
    print(a.__i)

main() # Call the main function
```

```
def main():
    a = A()
    a.print()

class A:
    def __init__(self, newS = "Welcome"):
        self.__s = newS

    def print(self):
        print(self.__s)

main() # Call the main function
```

3. Inheritance

① Superclass & Subclass

Superclass 父类 General

Subclass 子类 Specialized

inherit the DFs & Methods for superclass, and then specialize itself with more DFs and Methods

"is a"

GeometricObject

-color: str
-filled: bool

GeometricObject(color: str, filled: bool)
getColor(): str
setColor(color: str): None
isFilled(): bool
setFilled(filled: bool): None
__str__(): str

input output

The color of the object (default : green).	
Indicates whether the object is filled with a color (default: True).	
Creates a GeometricObject with the specified color and filled values.	
Returns the color.	
Sets a new color.	
Returrns the filled property.	
Sets a new filled property.	
Returns a string representation of this object.	

Circle

-radius: float

Circle(radius: float, color: str, filled: bool)
getRadius(): float
setRadius(radius: float): None
getArea(): float
getPerimeter(): float
getDiameter(): float
printCircle(): None

Rectangle

-width: float
-height: float

都是等数列和

Rectangle(width: float, height: float, color: string, filled: bool)
getWidth(): float
setWidth(width: float): None
getHeight(): float
setHeight(height: float): None
getArea(): float
getPerimeter(): float

开始一个object

```python
class GeometricObject:
    def __init__(self, color = "green", filled = True):
        self.__color = color
        self.__filled = filled

    def getColor(self):
        return self.__color

    def setColor(self, color):
        self.__color = color

    def isFilled(self):
        return self.__filled

    def setFilled(self, filled):
        self.__filled = filled

    def __str__(self):
        return "color: " + self.__color + \
            " and filled: " + str(self.__filled)
```

This method is used to output the data fields

class sub(super)

```python
from GeometricObject import GeometricObject
import math # math.pi is used in the class

class Circle(GeometricObject):
    def __init__(self, radius):
        super().__init__()
        self.__radius = radius

    def getRadius(self):
        return self.__radius

    def setRadius(self, radius):
        self.__radius = radius

    def getArea(self):
        return self.__radius * self.__radius * math.pi

    def getDiameter(self):
        return 2 * self.__radius

    def getPerimeter(self):
        return 2 * self.__radius * math.pi

    def printCircle(self):
        print(self.__str__() + " radius: " + str(self.__radius))
```

DF: Inherit manually

Method: Inherit automatically

Object class is the superclass of every class

`class ClassName:` `...`	Equivalent	`class ClassName(object):` `...`

__init__ & __str__ are both built-in methods of object class, they can run without calling

Actually, we just always override them

```
class A:
    def __init__(self, i = 0):
        self.i = i

    def m1(self):
        self.i += 1

    def __str__(self):
        return 'The content of this object is:' + str(self.i)

x = A(8)
print(x)
```

```
class A:
    def __init__(self,i=0):
        self.i=i

print(A.__str__(2))
```

[15] ✓ 0.3s

... 2

② Method overriding

Creating a method in subclass which comes from modification of the methods in superclass

```
class Circle(GeometricObject):
    # Other methods are omitted

    # Override the __str__ method defined in GeometricObject
    def __str__(self):
        return super().__str__() + " radius: " + str(radius)
```

③ Polymorphism

Object 1, Object 2, ..., Object n → The same function → Output 1, Output 2, ..., Output n

```
def main():
    # Display circle and rectangle properties

    c = Circle(4)
    r = Rectangle(1, 3)
    displayObject(c)
    displayObject(r)
    print("Are the circle and rectangle the same size?",
        isSameArea(c, r))

# Display geometric object properties
def displayObject(g):
    print(g.__str__())

# Compare the areas of two geometric objects
def isSameArea(g1, g2):
    return g1.getArea() == g2.getArea()

main() # Call the main function
```

Sometimes we may design polymophism wrongly:

```
def displayObject(g):
    print("Area is", g.getArea())          ✓
    print("Perimeter is", g.getPerimeter()) ✓
    print("Diameter is", g.getDiameter())   ✗
    print("Width is", g.getWidth())         ✗
    print("Height is", g.getHeight())       ✗
```

Using isinstance() function to solve this problem:

```
def main():
    # Display circle and rectangle properties
    c = Circle(4)
    r = Rectangle(1, 3)
    print("Circle...")
    displayObject(c)
    print("Rectangle...")
    displayObject(r)

# Display geometric object properties
def displayObject(g):
    print("Area is", g.getArea())
    print("Perimeter is", g.getPerimeter())

    if isinstance(g, Circle):
        print("Diameter is", g.getDiameter())
    elif isinstance(g, Rectangle):
        print("Width is", g.getWidth())
        print("Height is", g.getHeight())

main() # Call the main function
```

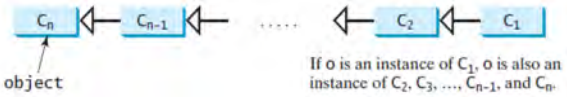
e.g.

a = 1.4

isinstance(a, float)

⇒ True

④ Dynamic binding



Descend

Invoke the methods

```
class C1:
    def __init__(self):
        self.f  = 1

    def output(self):
        print('In C1, the f is:',self.f)

class C2(C1):
    def __init__(self):
        self.f  = 2

    def output(self):
        print('In C2, the f is:',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()
```

```
class Student:
    def __str__(self):
        return "Student"

    def printStudent(self):
        print(self.__str__())

class GraduateStudent(Student):
    def __str__(self):
        return "Graduate Student"

a = Student()
b = GraduateStudent()
a.printStudent()
b.printStudent()
```

self 指向对象的 object

Practice

```
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()
```

P
S

(a)

```
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()
```

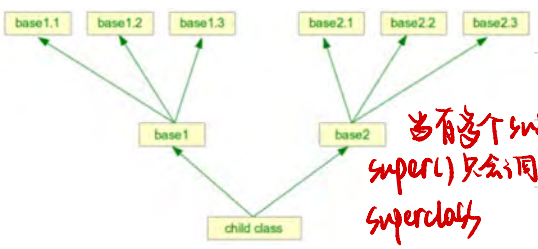
P
P

(b)

Coding in Brain

⇒ Understanding the flows carefully !

⑤ Multiple inheritance



Inheritance Tree

```
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):
        super().__init__(a)
        super().__init__(b)
        self.c=c

    def output(self):
        print(self.a)
        print(self.c)
        print(self.b)

def main():
    c = C(1,2,3)
    c.output()

main()
```

A()

当有多个superclass时，super()只会调用第一个superclass

2
3
Error

该程序

```
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)
        print(self.b)

def main():
    c = C(1,2,3)
    c.output()

main()
```

1
3
2

Lecture 8

DS: Stack. Queue. LinkedList. Tree

AR: Recursion. Binary Search. Sorting

1. Basic Understanding of Data Structure & Algorithm

① Definition

Data Structure: A systematic way to saving and using data

Algorithm: A step-by-step procedure for performing some task in a finite amount of time

② Goodness of algorithm

Running time ☆

Space usage

We can use "time( )" to roughly measure the running time of the code:

time( ) records the actual time of that moment in seconds

```
from time import time

startTime = time()

for i in range(1, 20000):
    if i%10 == 0:
        print(i)

endTime = time()

print('The time elapsed is:', endTime - startTime, 'seconds')
```

③ Algorithm analysis

a. Experimental analysis vs Theoretical analysis

Experimental analysis: ⓧ Hard to control the same environment (Hardware & Software)

Hard to do all experiments with different input sizes

Waste of time

Theoretical analysis: ⓥ

b. Three principals

operate each line is once primitive operation

⚠ Counting primitive operations (Operation time)

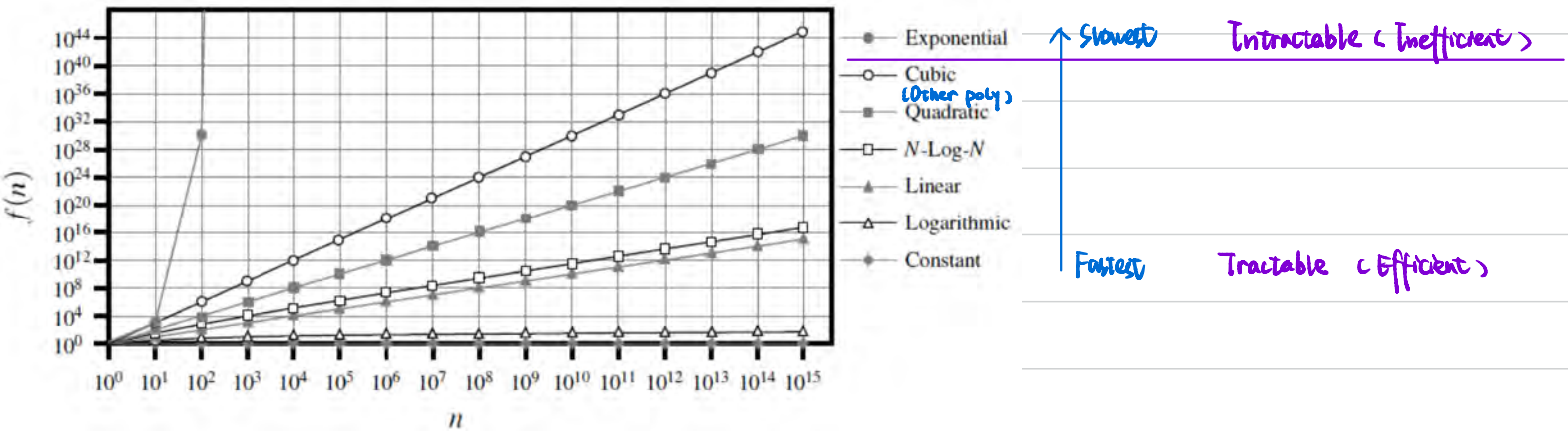
- We define a set of primitive operations such as the following:
  - ✓ Assigning a variable to an object
  - ✓ Determining the object associated with a variable
  - ✓ Performing an arithmetic operation (for example, adding two numbers)
  - ✓ Comparing two numbers
  - ✓ Accessing a single element of a Python list by index
  - ✓ Calling a function (excluding operations executed within the function) (Not executing)
  - ✓ Returning from a function.

⚠ Measuring operations as a function of input size

Operation ≙ Operation time

n: Input size

f(n): Operation time



(Other poly)

↑ Slowest — Intractable (Inefficient)

Fastest — Tractable (Efficient)

⇒ Asymptotic analysis (점근분석) → O(g(n))

$n \to \infty$

is Definition

f(n) is O(g(n)) if

$\exists\ c>0$ and $n_0 \geq 1$ s.t. $\forall\ n \geq n_0$, $f(n) \leq c\,g(n)$ (worst)

g(n) is actually an upper bound, the worst-case expectation

- Example: $5n^4 + 3n^3 + 2n^2 + 4n + 1$ is $O(n^4)$
- Example: 2^{n+2} is $O(2^n)$
- Example: $2n + 100log n$ is $O(n)$

Example: Finding the smallest number in a list

```
smallest_so_far = None
print('Before', smallest_so_far)

for num in [9, 39, 21, 98, 4, 5, 100, 65]:
    if smallest_so_far == None:
        smallest_so_far = num
    elif num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far, num)

print('After', smallest_so_far)
```

$3 + \max\{4, 5, 4\} \cdot n = 5n + 3$ (sample size에 비례)

⇓

O(n)

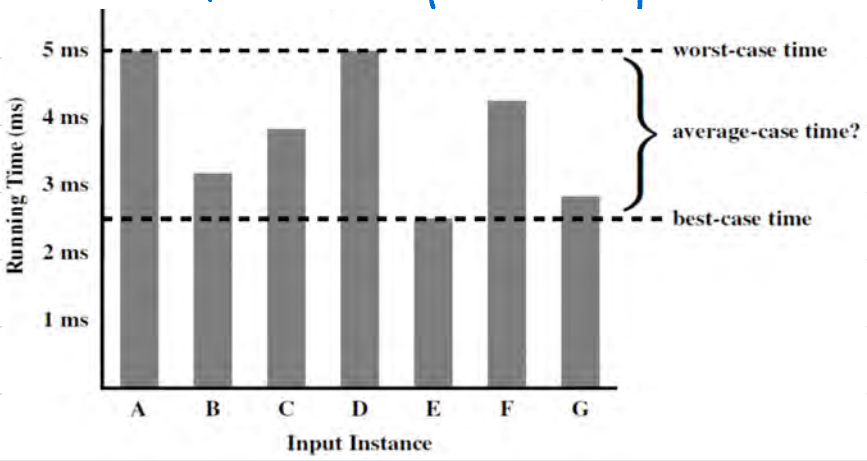
iii Comparison

If A is $O(n)$, and B is $O(n^2)$, then we say:

A is asymptotically better than B

②Focusing on the worst-case input

From worst-case input, we are able to find the problems of the codes



当worst case相同时，才比较average case

④Recursion

a. Definition

A function make calls to itself

```
def A():
```

A() is called when defining A()

b. Usage

① Define recursive definition

② Write the recursion codes

(With the help of recursive traces)

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 & \text{if } n \geq 1. \end{cases}$$

最后是一个 Fixed basic case

```
def facFunc(n):
    if n<0:
        print('Invalid input.')
        return None
    elif n == 0:
        return 1
    else:
        return n*facFunc(n-1)
```

$$power(x,n) = \begin{cases} 1 & \text{if } n = 0 \\ x \cdot \left(power\left(x, \lfloor \frac{n}{2} \rfloor\right)\right)^2 & \text{if } n > 0 \text{ is odd} \\ \left(power\left(x, \lfloor \frac{n}{2} \rfloor\right)\right)^2 & \text{if } n > 0 \text{ is even} \end{cases}$$

```
def myPower(x, n):
    if n==0:
        return 1
    else:
        partial = myPower(x, n//2)
        result = partial * partial
        if n%2==1:
            result = result * x
        return result
```

This one is better with $O(\log n)$: Using the Binary Search

要求：自次操作后即快分治减半

⑤ Binary Search

a. Process

前提

Sort the sequence ⇒ Low = 0, High = n-1, Mid = ½(L+H) ⇒ Case 1 Target = Mid / Case 2 Target > Mid / Case 3 Target < Mid ⇒ Narrow down to the half ⇒ Continue...

L/M/H 是 index

```
def binarySearch(data, target, low, high):
    if low>high:
        print('Cannot find the target number!')
        return False
    else:
        mid = (low+high)//2
        if target==data[mid]:
            print('The target number is at position',mid)
            return True
        elif target<data[mid]:
            return binarySearch(data, target, low, mid-1)
        else:
            return binarySearch(data, target, mid+1, high)

def main():
    data = [1, 3, 5, 6, 16, 78, 100, 135, 900]
    target = 16
    binarySearch(data, target, 0, len(data)-1)
```

不用特別取 Low & High

b. Complexity

- Sequential Search : O(n)
- Binary Search: O(log n)

⇓ Why?

$\frac{n}{2^1} \Rightarrow \frac{n}{2^2} \Rightarrow \cdots \Rightarrow \frac{n}{2^d} = 1$ (Stop!)

⇓

$d = \log_2 n$

Lecture 9

1. More about Recursion

① Types

Linear Recursion : Call itself once

Multiple Recursion : Call itself more than once

② Linear Recursion

Compute the sum of a list:

```
def linearSum(L, n):
    if n==0:
        return 0
    else:
        return linearSum(L, n-1)+L[n-1]

def main():
    L = [1, 2, 3, 4, 5, 9, 100, 46, 7]
    print('The sum is:', linearSum(L, len(L)))
```

③ Multiple Recursion

Compute the sum of a list by Binary sum.

```
def binarySum(L, start, stop):
    if start>=stop:
        return 0
    elif start==stop - 1:
        return L[start]
    else:
        mid = (start+stop)//2
        return binarySum(L, start, mid)+binarySum(L, mid, stop)

def main():
    L = [1, 2, 3, 4, 5, 6, 7]
    print(binarySum(L, 0, len(L)))
```

求Floor function

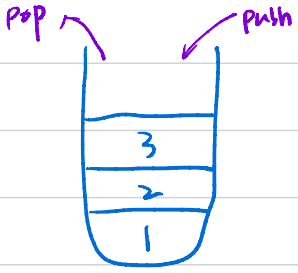
最后是7个BinarySum的和

0)

[0,3) [3,7)

[0,1)[1,3)[3,5)[5,7)

此时只剩一个元素 ⇒ 输出!



2. Stack

① Definition

Data structure which comply with First-in, Last-out principle.

② The stack class

We want to define stack as a class then reuse it as one kind of Data Structure

S.push(e): Add element e to the top of stack S.

S.pop(): Remove and return the top element from the stack S; an error occurs if the stack is empty.

S.top(): Return a reference to the top element of stack S, without removing it; an error occurs if the stack is empty.

S.is_empty(): Return True if stack S does not contain any elements.

len(S): Return the number of elements in stack S; in Python, we implement this with the special method __len__.

```
class ListStack:

    def __init__(self):
        self.__data = list()

    def __len__(self):
        return len(self.__data)

    def is_empty(self):
        return len(self.__data) == 0

    def push(self, e):
        self.__data.append(e)

    def top(self):
        if self.is_empty():
            print('The stack is empty.')
        else:
            return self.__data[self.__len__()-1]

    def pop(self):
        if self.is_empty():
            print('The stack is empty.')
        else:
            return self.__data.pop()
```

因为list和stack的原始方法都有(append(), pop()),

故可以直接用list来实现stack class

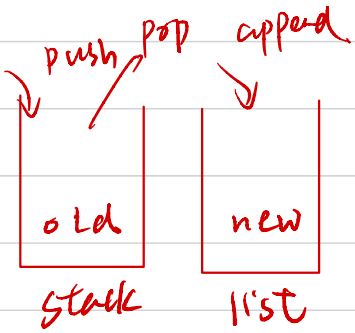
List: 可以access任意元素

Stack: 只能access top元素

③Some applications of stack

a. Reverse a list

```
from stack import ListStack

def reverse_data(oldList):
    s = ListStack()
    newList = list()

    for i in oldList:
        s.push(i)

    while (not s.is_empty()):
        mid = s.pop()
        newList.append(mid)

    return newList

def main():
    oldList = [1,2,3,4,5]
    newList = reverse_data(oldList)
    print(newList)
```



b. Brackets match checking

```
from stack import ListStack

def is_matched(expr):
    lefty = '([{'
    righty = ')]}'

    s = ListStack()

    for c in expr:

        if c in lefty:
            s.push(c)
        elif c in righty:
            if s.is_empty():
                return False
            if righty.index(c)!=lefty.index(s.pop()):
                return False
    return s.is_empty()

def main():
    expr = '1+2*(3+4)-[5-6]'
    print(is_matched(expr))
    expr = '((())}]'
    print(is_matched(expr))
```

C. Matching tags in HTML Language

检查头指令和尾指令是否对应

- body: document body
- h1: section header
- center: center justify
- p: paragraph
- ol: numbered (ordered) list
- li: list item

```
<body>
<center>
<h1> The Little Boat </h1>
</center>
<p> The storm tossed the little
boat like a cheap sneaker in an
old washing machine.  The three
drunken fishermen were used to
such treatment, of course, but
not the tree salesman, who even as
a stowaway now felt that he
had overpaid for the voyage. </p>
<ol>
<li> Will the salesman die? </li>
<li> What color is the boat? </li>
<li> And what about Naomi? </li>
</ol>
</body>
```

(a)

The Little Boat

The storm tossed the little boat like a cheap sneaker in an old washing machine. The three drunken fishermen were used to such treatment, of course, but not the tree salesman, who even as a stowaway now felt that he had overpaid for the voyage.

1. Will the salesman die?
2. What color is the boat?
3. And what about Naomi?

(b)

```python
from stack import ListStack

def is_matched_html(raw):
    s = ListStack()
    j = raw.find('<')

    while j!=-1:
        k = raw.find('>', j+1)
        if k==-1:
            return False
        tag = raw[j+1:k]

        if not tag.startswith('/'):
            s.push(tag)
        else:
            if s.is_empty():
                return False
            if tag[1:]!=s.pop():
                return False
        j = raw.find('<', k+1)

    return s.is_empty()

def main():
    fhand = open('sampleHTML.txt', 'r')
    raw = fhand.read()
    print(raw)
    print(is_matched_html(raw))
```

这部分就是 <~>

"raw" 是 a string

3. Queue

① Definition

Data structure which comply with First-in, First out principle.

② The queue class

Q.enqueue(e): Add element e to the back of queue Q.

Q.dequeue( ): Remove and return the first element from queue Q; an error occurs if the queue is empty.

Q.first( ): Return a reference to the element at the front of queue Q, without removing it; an error occurs if the queue is empty.

Q.is_empty( ): Return True if queue Q does not contain any elements.

len(Q): Return the number of elements in queue Q; in Python, we implement this with the special method __len__.

加之前检查是否已满
减之前检查是否已空

这两个名字与stack的一致

Queue的Data Field 也是个List

```python
class ListQueue:
    default_capacity = 5

    def __init__(self):
        self.__data = [None]*ListQueue.default_capacity
        self.__size = 0
        self.__front = 0
        self.__end = 0

    def __len__(self):
        return self.__size

    def is_empty(self):
        return self.__size ==0

    def first(self):
        if self.is_empty():
            print('Queue is empty.')
        else:
            return self.__data[self.__front]

    def dequeue(self):

        if self.is_empty():
            print('Queue is empty.')
            return None

        answer = self.__data[self.__front]
        self.__data[self.__front] = None
        self.__front = (self.__front+1) \
                       % ListQueue.default_capacity
        self.__size -=1
        return answer

    def enqueue(self, e):
        if self.__size == ListQueue.default_capacity:
            print('The queue is full.')
            return None

        self.__data[self.__end] = e
        self.__end = (self.__end+1) \
                     % ListQueue.default_capacity
        self.__size += 1

    def outputQ(self):
        print(self.__data)
```

此时我们不用 append | pop，因为这两个 method 只能处理 list 的尾部，因此，我们要在 __init__() 定义一个size (Stack 尾 / Queue 头)

③ Application

Simulating a web service

```
from ListQueue import ListQueue
from random import random
from math import floor

class WebService():
    default_capacity = 5
    def __init__(self):
        self.nameQ = ListQueue()
        self.timeQ = ListQueue()

    def taskArrive(self,taskName,taskTime):
        if self.nameQ.__len__() < WebService.default_capacity:
            self.nameQ.enqueue(taskName)
            self.timeQ.enqueue(taskTime)
            print('A new task 《'+taskName+'》 has arrived and is waiting for processing...')
        else:
            print('The service queue of our website is full, the new task is dropped.')

    def taskProcess(self):
        if (self.nameQ.is_empty() == False):
            taskName = self.nameQ.dequeue()
            taskTime = self.timeQ.dequeue()
            print('Task 《'+taskName+'》 has been processed, it costs '+str(taskTime)+' seconds.')


def main():
    ws = WebService()
    taskNameList = ['Dark knight','X-man','Kungfu','Shaolin Soccer','Matrix','Walking in the clouds'\
                    ,'Casino Royale','Bourne Supremacy','Inception','The Shawshank Redemption']

    print('Simulation starts...')
    print('-----------------------------------')
    for i in range(1,31):
        rNum = random()
        if rNum<=0.6:
            taskIndex = floor(random()*10)
            taskTime  = floor(random()*1000)/100
            ws.taskArrive(taskNameList[taskIndex],taskTime)
        else:
            ws.taskProcess()
    print('-----------------------------------')
    print('Simulation finished.')
```

模拟30次

[0,1)

0.6的概率在添加

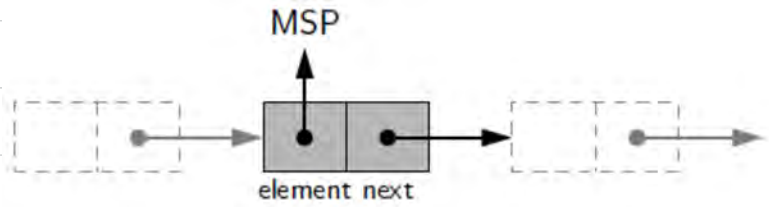
0.4的概率在被叫

Lecture 10

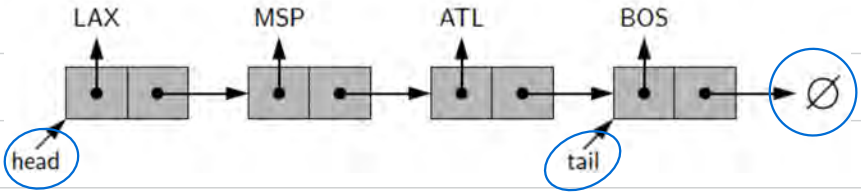
1. Singly Linked list 单链表

① Definition

A collection of nodes



每个Node包含两部分 { Reference to an object
Reference to the next node



(最后指向None)

无法access哪个node，都要从head开始 ⟶ (除了H&T)

a. Create a linked list

先定义Node

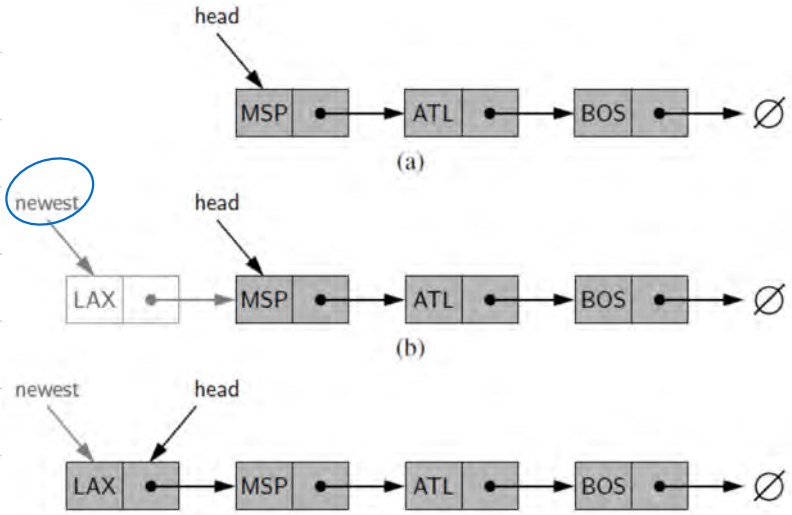
```
class Node:
    def __init__(self, element, pointer):
        self.element = element
        self.pointer = pointer

class LinkedList:

    def __init__(self):
        self.head = None
        self.tail = None
        self.size = 0
```

后定义LinkedList

```
class Node:
    def __init__(self, element, pointer):
        self.element = element
        self.pointer = pointer

class LinkedList:

    def __init__(self):
        self.head = None
        self.tail = None
        self.size = 0

    def add_first(self, e):
        newest = Node(e, None)
        newest.pointer = self.head
        self.head = newest
        self.size = self.size+1

        if self.size == 1:
            self.tail = newest

    def add_last(self, e):
        newest = Node(e, None)
        if self.size > 0:
            self.tail.pointer = newest
        else:
            self.head = newest
        self.tail = newest
        self.size = self.size+1
```

```
def remove_first(self):
    if self.size == 0:
        print('The linked list is empty')
    elif self.size == 1:
        answer = self.head.element
        self.head = None
        self.tail = None
        self.size -= 1
        return answer
    else:
        answer = self.head.element
        self.head = self.head.pointer
        self.size = self.size - 1
        return answer
```

b. Insert an element at the head

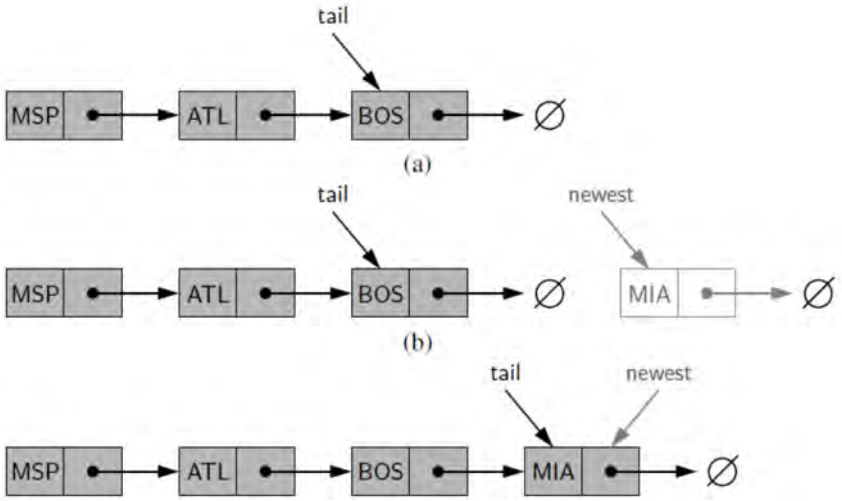


(i) 创建一个新的 Node
(ii) 令新的 Node 指向原来的 head
(iii) 令 head pointer 指向新的 Node
(iv) size 变化

```
class Node:
    def __init__(self, element, pointer):
        self.element = element
        self.pointer = pointer

class LinkedList:

    def __init__(self):
        self.head = None
        self.tail = None
        self.size = 0
```

```
def add_first(self, e):
    (i) newest = Node(e, None)
    (ii) newest.pointer = self.head
    (iii) self.head = newest
    (iv) self.size = self.size+1

    if self.size == 1:
        self.tail = newest
```

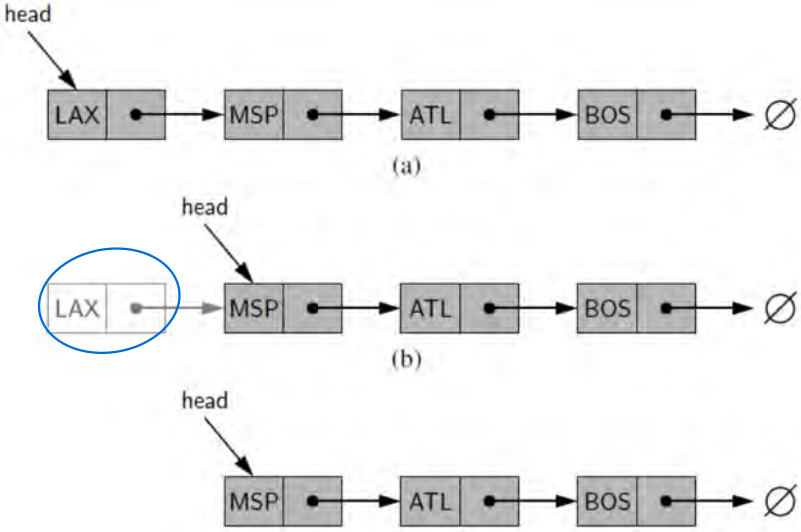
c. Insert an element at the tail



(i) 创建一个新的 Node
~~(ii) 令新的 Node 指向 ∅~~ 实际上不用
(iii) 令原来的 tail 指向新的 Node
(iv) 令 tail pointer 指向新的 Node
(v) size 变化

```
def add_last(self, e):
    (i) newest = Node(e, None)
    if self.size > 0:
        (iii) self.tail.pointer = newest
    else:
        self.head = newest
    (iv) self.tail = newest
    (v) self.size = self.size+1
```

d. Remove an element at the head



以令 head pointer 指向下一个Node

(原来和Node因为没有被指，所以会消失)

```
def remove_first(self):
    if self.size == 0:
        print('The linked list is empty')
    elif self.size == 1:
        answer = self.head.element
        self.head = None
        self.tail = None
        self.size -= 1
        return answer
    else:
        answer = self.head.element
        self.head = self.head.pointer
        self.size = self.size - 1
        return answer
```

分3种情况讨论

直接设置self.head

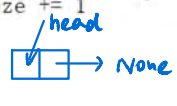
② Practices

a. LinkedStack

Practice: Implement stack with a singly linked list

```
class Node:
    def __init__(self, element, pointer):
        self.element = element
        self.pointer = pointer

class LinkedStack:

    def __init__(self):
        self.head = None
        self.size = 0

    def __len__(self):
        return self.size

    def is_empty(self):
        return self.size == 0

    def push(self,e):
        self.head = Node(e, self.head)
        self.size += 1
```

这里做了什么 (add_first!!)




```
def top(self):
    if self.is_empty():
        print('Stack is empty.')
    else:
        return self.head.element

def pop(self):
    if self.is_empty():
        print('Stack is empty.')
    else:
        answer = self.head.element
        self.head = self.head.pointer
        self.size -=1
        return answer
```

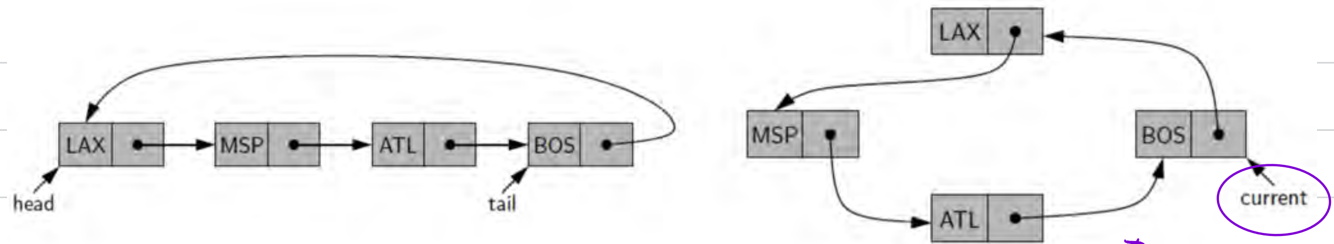
b. Linkedqueue

Practice: Implement queue with a singly linked list

```
class LinkedQueue:

    def __init__(self):
        self. head = None
        self. tail = None
        self. size = 0

    def __len__(self):
        return self.size

    def is_empty(self):
        return self.size == 0

    def first(self):
        if self.is_empty():
            print('Queue is empty.')
        else:
            return self.head.element

    def dequeue(self):
        if self.is_empty():
            print('Queue is empty.')
        else:
            answer = self.head.element
            self.head = self.head.pointer
            self.size -= 1
            if self.is_empty():
                self.tail = None
            return answer

    def enqueue(self, e):
        newest = Node(e, None)

        if self.is_empty():
            self.head = newest
        else:
            self.tail.pointer = newest
        self.tail = newest
        self.size += 1
```

2. Circularly Linked list 循环链表

① Definition



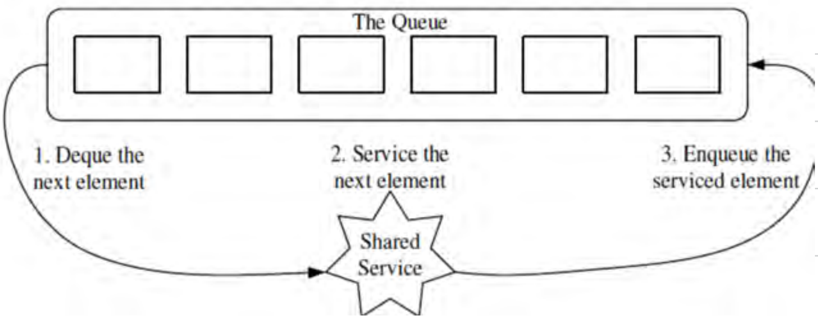
在circularly LL中，我们只需一个"current"

将tail的pointer设为head，就可以把singly linked list 转化为 circularly linked list

② Practice

Round-robin scheduler 循环排程表 cpu每个时间区段内只运行单项任务

a. Method 1: Queue

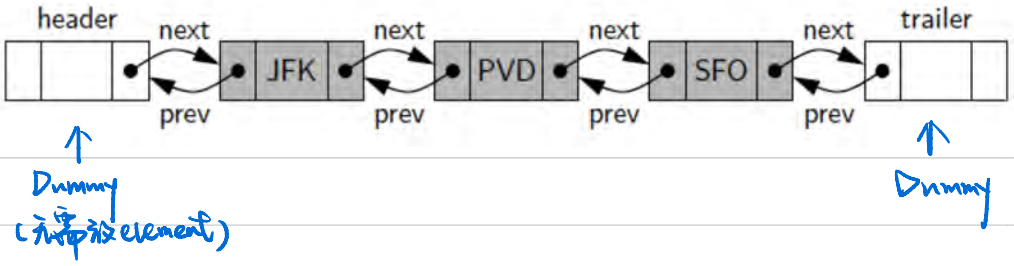


b. Method 2 : Queue + Circularly Linked list (Faster)

```
class Node:
    def __init__(self, element, pointer):
        self.element = element
        self.pointer = pointer

class CQueue:

    def __init__(self):
        self.__tail = None
        self.__size = 0

    def __len__(self):
        return self.__size

    def is_empty(self):
        return self.__size == 0

    def first(self):

        if self.is_empty():
            print('Queue is empty.')
        else:
            head = self.__tail.pointer
            return head.element
    def dequeue(self):
        if self.is_empty():
            print('Queue is empty.')
        else:
            oldhead = self.__tail.pointer
            if self.__size == 1:
                self.__tail = None
            else:
                self.__tail.pointer = oldhead.pointer
            self.__size -= 1
            return oldhead.element

    def enqueue(self,e):
        newest = Node(e,None)
        if self.is_empty():
            newest.pointer = newest
        else:
            newest.pointer = self.__tail.pointer
            self.__tail.pointer = newest
        self.__tail = newest
        self.__size += 1
```

3. Doubly linked list

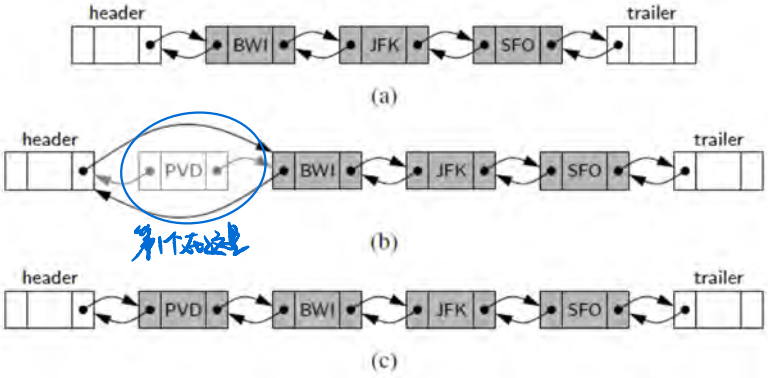
① Definition



Dummy (不存放element)

Dummy

a. Insert an element at the head



第1个在这里

在后端增加：O(C)

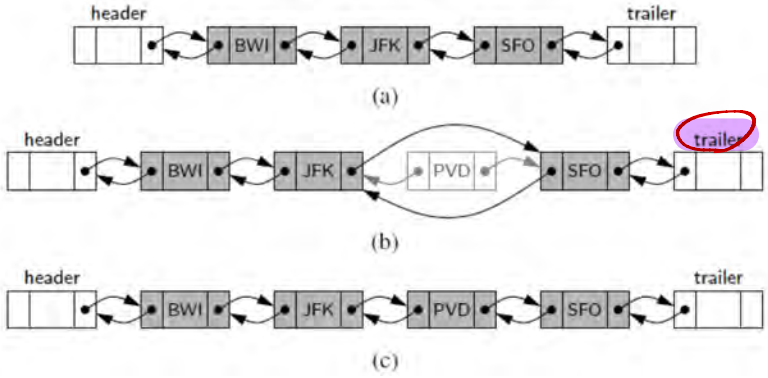
在中间加：O(n)

① 创建一个新的Node

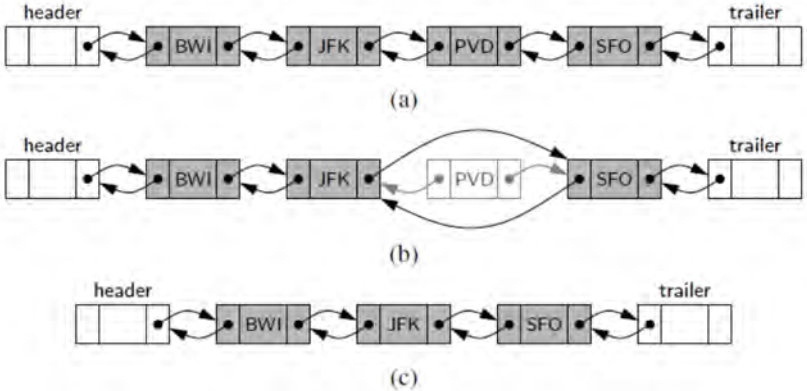
② 令 Prev 指向 header，Next 指向原第1个 Node

③ 令 Header 的 Next 和原第1个 Node 的 Prev 都指向新的 Node

b. Insert an element in the middle



c. Delete an element in the middle




```
class Node:
    def __init__(self, element, prev, nxt):
        self.element = element
        self.prev = prev
        self.nxt = nxt

class DLList:

    def __init__(self):
        self.header = Node(None, None, None)
        self.trailer = Node(None, None, None)
        self.header.nxt = self.trailer
        self.trailer.prev = self.header
        self.size = 0

    def __len__(self):
        return self.size

    def is_empty(self):
        return self.size == 0
    def insert_between(self, e, predecessor, successor):
        newest = Node(e, predecessor, successor)
        predecessor.nxt = newest
        successor.prev = newest
        self.size+=1
        return newest

    def delete_node(self, node):
        predecessor = node.prev
        successor = node.nxt
        predecessor.nxt = successor
        successor.prev = predecessor
        self.size -=1
        element = node.element
        node.prev = node.nxt = node.element = None
        return element

    def iterate(self):
        pointer = self.header.nxt
        print('The elements in the list:')
        while pointer != self.trailer:
            print(pointer.element)
            pointer = pointer.nxt
def main():
    d=DLList()
    d.__len__()

    newNode = d.insert_between(10, d.header, d.trailer)
    newNode = d.insert_between(20, newNode, d.trailer)
    newNode = d.insert_between(30, newNode, d.trailer)
    d.iterate()
    d.delete_node(d.header.nxt.nxt)
    d.iterate()
```

4. Sorting

① Bubble sort 冒泡排序

a. Process

1) Iterate over a list of numbers, compare every element i with the following element i+1, and swap them if i is larger
2) Iterate over the list again and repeat the procedure in step 1, but ignore the last element in the list
3) Continuously iterate over the list, but each time ignore one more element at the tail of the list, until there is only one element left

O: $(n-1)+(n-2)+\cdots+1=(n-1)\frac{n}{2} \Rightarrow O(n^2)$

b. Practice

⚠ Bubble sort over a standard list

```
def bubble(bubbleList):
    listLength = len(bubbleList)
    while listLength > 0:
        for i in range(listLength - 1):
            if bubbleList[i] > bubbleList[i+1]:
                buf = bubbleList[i]
                bubbleList[i] = bubbleList[i+1]
                bubbleList[i+1] = buf
        listLength -= 1
    return bubbleList


def main():
    bubbleList = [3, 4, 1, 2, 5, 8, 0, 100, 17]
    print(bubble(bubbleList))
```

⚠ Bubble sort over a singly linked list

```
from LinkedQueue import LinkedQueue

def LinkedBubble(q):
    listLength = q.size

    while listLength > 0:
        index = 0
        pointer = q.head
        while index < listLength-1:
            if pointer.element > pointer.pointer.element:
                buf = pointer.element
                pointer.element = pointer.pointer.element
                pointer.pointer.element = buf
            index += 1
            pointer = pointer.pointer
        listLength -= 1
    return q
```

```
def outputQ(q):
    pointer = q.head

    while pointer:
        print(pointer.element)
        pointer = pointer.pointer

def main():
    oldList = [9, 8, 6, 10, 45, 67, 21, 1]
    q = LinkedQueue()

    for i in oldList:
        q.enqueue(i)

    print('Before the sorting...')
    outputQ(q)

    q = LinkedBubble(q)
    print()
    print('After the sorting...')
    outputQ(q)
```

② Quick sort

a. Process

1) Pick an element, called a pivot, from the array
2) Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation
3) Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values

每次选pivot，都使pivot在正确的位置

Worst case: $O(n^2)$

Average case: $O(n \log n)$

b. Practice

Quick sort over a standard list

```
def quickSort(L, low, high):
    i = low
    j = high
    if i>=j:
        return L
    pivot = L[i]

    while i<j:
        while i<j and L[j]>=pivot:
            j = j - 1
        L[i] = L[j]

        while i<j and L[i]<=pivot:
            i = i + 1
        L[j] = L[i]
    L[i] = pivot

    quickSort(L, low, i-1)
    quickSort(L, j+1, high)
    return L

L = [6, 5, 3, 10, 12, 2, 4]
print(quickSort(L, 0, 6))
```

Lecture 11

1. Tree

① Basics

- Parent & Children — 父若有，则只有一个
- Root
- Leaf node — 没有 children 则称其为 leaf node（不一定在最下一层）
- Internal node — 若有至少一个 child，则称其为 internal node
- Edge — each pair of nodes
- Path
- Depth — length of path to root（# of edges）
- Subtree — left subtree / right subtree 具有递归的特点
- Proper / Full BT — # of children of each node = 0 / 2

② Binary tree

a. Properties:

- At most 2 children
- Be labeled as a left child / right child
- Ordered tree

b. Define a tree class

```
class Node:

    def __init__(self, element, parent = None, \
        left = None, right = None):
        self.element = element
        self.parent = parent
        self.left = left
        self.right = right

class LBTree:

    def __init__(self):
        self.root = None
        self.size = 0

    def __len__(self):
        return self.size

    def find_root(self):
        return self.root

    def parent(self, p):
        return p.parent

    def left(self, p):
        return p.left

    def right(self, p):
        return p.right

    def num_child(self, p):
        count = 0
        if p.left is not None:
            count+=1
        if p.right is not None:
            count+=1
        return count
```

```
    def add_root(self, e):
        if self.root is not None:
            print('Root already exists.')
            return None
        self.size = 1
        self.root = Node(e)
        return self.root

    def add_left(self, p, e):
        if p.left is not None:
            print('Left child already exists.')
            return None
        self.size+=1
        p.left = Node(e, p)
        return p.left
    def add_right(self, p, e):
        if p.right is not None:
            print('Right child already exists.')
            return None
        self.size+=1
        p.right = Node(e, p)
        return p.right

    def replace(self, p, e):
        old = p.element
        p.element = e
        return old

    def delete(self, p):
        if p.parent.left is p:
            p.parent.left = None
        if p.parent.right is p:
            p.parent.right = None
        return p.element
```

```
def main():
    t = LBTree()
    t.add_root(10)
    t.add_left(t.root, 20)
    t.add_right(t.root, 30)
    t.add_left(t.root.left, 40)
    t.add_right(t.root.left, 50)
    t.add_left(t.root.right, 60)
    t.add_right(t.root.left.left, 70)

    print(t.root.element)
    print(t.root.left.element)
    print(t.root.right.element)
    print(t.root.left.right.element)
```

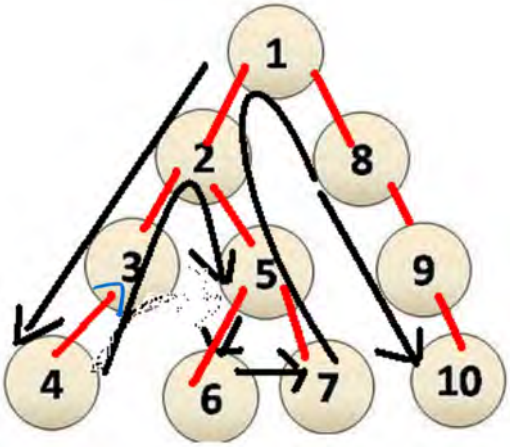
```
>>> main()
10
20
30
50
```

③ Search

a. DFS

△ Definition

Three principles: Root first / 左先右后 / 深度优先

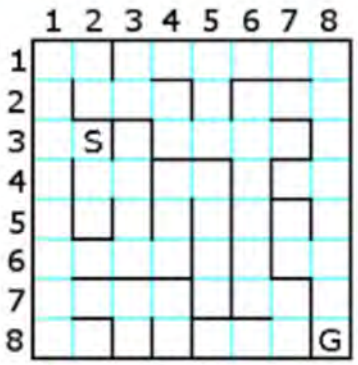


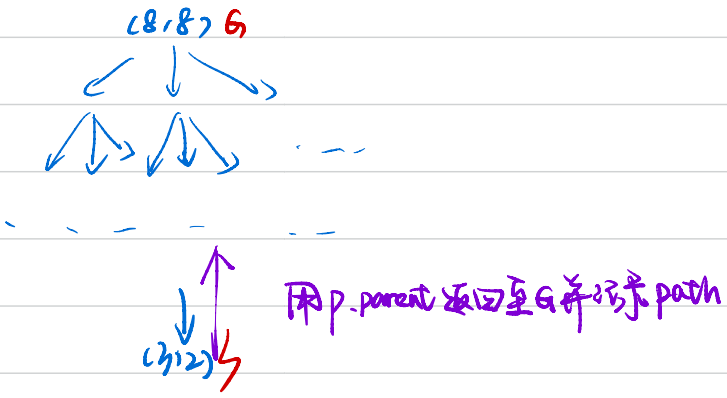
1 2 3 4 3 2 5 6 5 7 5 2 1 8 9 10

```
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 is not None:
            DFSearch(t.right)
```

△ Example

Example: search a path in a maze

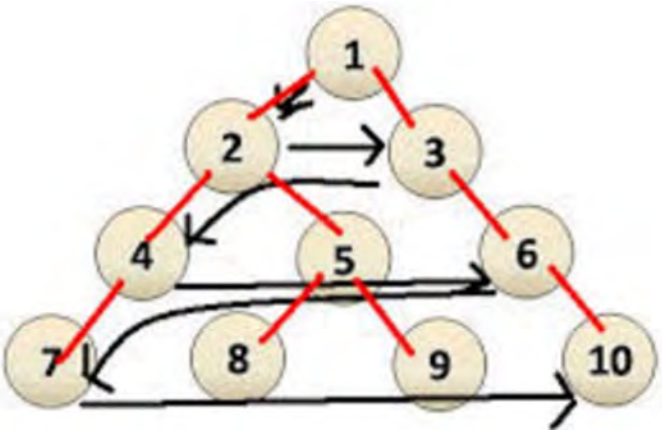




b. BFS

△ Definition

Three principles: Root first / 左先右后 / 广度优先（level 由高到低）

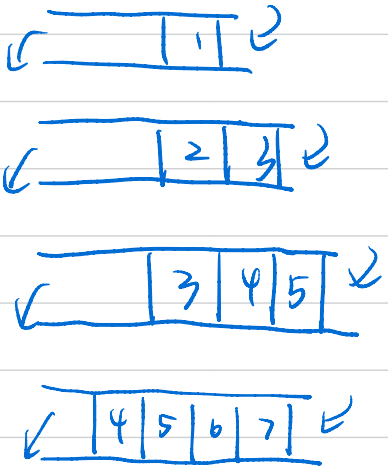


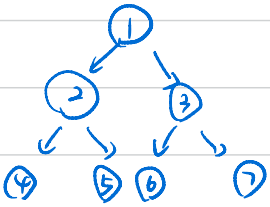
Level 0

1 2 3 4 5 6 7 8 9 10

Level 4

```
def BFSearch(t):

    q = ListQueue()
    q.enqueue(t)

    while q.is_empty() is False:
        cNode = q.dequeue()
        if cNode.left is not None:
            q.enqueue(cNode.left)
        if cNode.right is not None:
            q.enqueue(cNode.right)
        print(cNode.element)
```





左儿子先进，右儿子后进