# 数据类型

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | C | JAVA | LUA | PYTHON | PHP | R | JS |
| 数字 |  |  | 123 | 123 | 123 | 123 | 123 |
| 整数和浮点小数转换 |  |  |  | Int和float |  |  |  |
| 数字按一定小数位输出 |  |  | print(string.format("%.yf",  xx.xxxx)) | (1)print(round(xx.xxx)  ,y)（会四舍五入）  (2)print(“%.yf”%xx.xxx) |  |  |  |
| 字符串 |  |  | “123” or abc | “123” or abc | “123” or abc | “123” or abc | “123” or abc |
| 字符串和数字转换 |  |  | tonumber和tostring | Int和str |  |  |  |
| 字符串长度 |  |  |  | len(a)用于求字符串长度 | strlen(a)用于求字符串长度 |  |  |
| 删除字符串末尾空白 |  |  |  | a = a.rstrip() |  |  |  |
| 字符串按符号拆分 |  |  | 拆分：Split(a,”xx”)  xx表示拆分符号 | 拆分：a.split(”xx”)  xx表示拆分符号 | 拆分：explode(“xx”,$a) |  |  |
| 字符串拼接 |  |  | ..（两点） | +（加号） | .（一点） | 不太需要 | +（加号） |
| 字符串按字节切分 |  |  | a=”123bc”  a:sub(start,end)  a:sub(start,-1) 至末尾 | a=”123bc”  a[start:end+1:length]  a[start: ] 至末尾 | $a=”123bc”  substr($a,start,length) |  |  |
| 判断某一字符是否在字符串中 |  |  | If a:find(xx)~=nil | If a.find(xx)~=-1 |  |  |  |
| 常见转义字符串 |  |  | [1]Tab: “\t”  [2]换行: “\n” | [1]Tab: “\t”  [2]换行: “\n” | [1]Tab: “\t”  [2]换行: “<br/>”(浏览器输出)或”\n”(先用<pre>定义) | 不太需要 | [1]Tab: “\t”  [2]换行: “<br/>”或”<br>”(浏览器输出) |
| 布尔值 |  |  | false, true | False, True | False true | false true | False true |
| 空值 |  |  | nil (==false) | None | NULL (==false) | NA (==false) | NULL (==false) |
| 判断某一对象是否为空值 |  |  | If a==nil then | If a==None:  或者  If not a: |  |  |  |

\*补充：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | C | JAVA | LUA | PYTHON | PHP | R | JS |
| 数据类型是否需要预先声明 |  |  | 不需要 | 不需要 | 不需要 | 不需要 | 不需要 |
| 默认的局部变量 |  |  | 1.For 循环中自增（减）的i,j,k  2.函数内部变量 | 函数内部变量 | 函数内部变量 | 函数内部变量 | 函数内部变量 |
| 默认数字精度 |  |  | 双精度浮点数(14个小数位) | 双精度浮点数(17个小数位) | 双精度浮点数  (14个小数位) | 单精度浮点数(7个小数位) | 双精度浮点数  (17个小数位) |

补充：变量、变量值，地址、指针、索引号的联系和区别：

（1）变量，变量值和地址：变量值是变量的数据具体化，其在内存中的存放位置即为地址。地址是变量在计算机中的物理别名，两者是完全等价的。如下：

import ctypes

a=3 #a为变量，3为变量值

address\_a=id(a) #获取变量的内存地址

print(address\_a)

print(ctypes.cast(address\_a,ctypes.py\_object).value) #由地址来反求出变量值

140733696558816

3

（2）内存地址和指针：指针是用来保存变量内存地址数字的另一变量，简单情况下可理解为上面的address。需注意，由于指针也是变量，故它也有地址！

（3）内存地址和索引号：索引号是向量（数组）中用于索引元素的位置标记，显然它并不是向量（数组）中元素或变量的真实地址，而仅仅是抽象意义上的地址，但某种意义上两者可以等价看待，如下：

import ctypes

c=[1,2,3,4]

cc=[id(c[0]),id(c[1]),id(c[2]),id(c[3])]

for i in range(len(cc)):

    print(ctypes.cast(cc[i],ctypes.py\_object).value)

1

2

3

4

# 基础数据结构

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | C | | JAVA | LUA | | PYTHON | PHP | R | | JS |
| 向量（数组） |  | |  | {1,2,3} | | [1,2,3] | [1,2,3] | c(1,2,3) | | [1,2,3] |
| 向量定义 |  | |  | B={1,2,3} | | B=[1,2,3] | $B=[1,2,3]  $B=array(1,2,3) | B=c(1,2,3) | | B=[1,2,3]  B=array(1,2,3) |
| 向量长度 |  | |  | #B | | len(B) | count($B) | length(B) | | B.length |
| 向量拼接 |  | |  |  | | A+B |  |  | |  |
| 向量元素排序 |  | |  | 正序：Table.sort(B)  逆序： | | 正序：B.sort()  逆序：B.reverse() |  | Sort(B) | |  |
| 向量元素按符号拼接 |  | |  | 拼接：table.concat(B,  ”xx”) xx表示拼接符号 | | 拼接：concat(B,  ”xx”) xx表示拼接符号 |  |  | |  |
| 向量内插入元素，删除元素，修改元素 |  |  | | | 插入：table.insert(B,sp, xxx)  删除：  table.remove(B,sp)  修改：  B[sp]=xx | 插入：B.insert(sp, xxx)  删除：del A[sp]  或A.pop(sp)（删除值可返回）  修改：A[sp]=xx |  | |  |  |
| 向量入栈与出栈 |  | |  | 入栈：B[#B+1]=xx  出栈：B[#B]=nil | | 入栈：B.append(xx)  出栈：B.pop() | 入栈：B[count($B)+1]=xx  出栈：  B[count($B)]=NULL |  | |  |
| 向量按“索引号”切分 |  | |  |  | | 切分：B[start:end+1]  \*全复制：B[:] | B[1:5] |  | |  |
| 判断元素是否在向量（字典）中 |  | |  | If A[i] then | | If xx in A: |  |  | |  |
| 矩阵 |  | |  | {{1,2},{3,4},{5,6}}  B[2][1]=3 | | [[1,2],[3,4],[5,6]]  B[2][1]=3 | [[1,2],[3,4],[5,6]]  $B[2][1]=3 | c((1,2),(3,4),(5,6))  注意：B[2.1]=3 | | [[1,2],[3,4],[5,6]]  B[2][1]=3 |
| 矩阵定义 |  | |  | B[i]={}  B={{1,2},{3,4},{5,6}} | | B[i]=[]  B=[[1,2],[3,4],[5,6]] | $B[i]=[]  $B=[[1,2],[3,4],[5,6]] | C=c(1,2,3,4,5,6)  B=matrix(C,3,2) | | B[i]=[]  B=[[1,2],[3,4],[5,6]] |
| 字典（散列表）定义 |  | |  | B={[“a”]=1,[“b”]=2} | | B={“a”:1,”b”:2} | $B=array(“a”=>1,  “b”=>2) | B=c("a"=1,"b"=2) | | B=array(“a”=>1,  “b”=>2) |
| 字典添加，修改，删除键值对 |  | |  | 添加：B[“c”]=3  修改：B[“c”]=4  删除：B[“c]=nil | | 添加：B[“c”]=3  修改：B[“c”]=4  删除: del B[“c”] |  |  | |  |
| 其他 |  | |  | 向量索引号从1开始 | | A=[1,2,3]，可知A中预先并不存在A[3]，那么A[3]就没有任何意义，则无论对A[3]赋值（A[3]=xx）还是输出A[3]均会报错！  Python中集合定义方式为：B={1,2,3}，特殊在于集合中的重复元素无意义。集合支持列表函数并且具备自身的集合运算函数，如 A |B 表示集合A，B的并集；A & B表示交集； A – B表示差集。  python严格区分列表，集合和字典的定义方式 |  | 还有诸如数据框，列表等独特结构  向量索引号从1开始 | |  |

\*补充：数据的物理结构和逻辑结构

物理结构：数据在计算机中的真实存储方式

（1）顺序结构：各数据的在内存中的地址（可简单理解为索引号）是“相邻”（隔4或8个字节）且连续的。例如顺序列表

（2）链式结构：各数据（节点）在内存中的地址是任意的，但对任一节点，其地址仅仅取决于前（后）一个节点的指针。例如链式列表

（3）散列结构：各数据在内存中的地址通过散列函数，实现与其索引号（关键字）的一一对应，但各元素地址间并无直接联系。如散列表

逻辑结构：数据在程序或算法中的抽象关系

（1）集合：各数据（元素）同属于一个集合体，但彼此之间并无关系，简单来说就是各数据之间无先后的序关系。如字典

（2）线性表：除头元素和尾元素之外，各元素有且仅有一个前驱元素（上一位置或节点）和一个后继元素（下一位置或节点）。例如 列表，栈，队列，链表

（3）非线性表：各元素可以有一个或多个前驱元素和后继元素。如二叉树，图

\*线性表和非线性表均可由顺序结构和链式结构构造

特别地，对于线性表来说，当元素为数字时，还可以根据数的序关系分为有序（升序，降序）表和无序表，有序列表对于后续算法有巨大影响。

\*补充：链式结构（以单向链表为例）

# 定义节点（元素）类

class Node:

    def \_\_init\_\_(self,data,next=None):

        self.data=data

        self.next=next

# 定义链表类

class LinkList:

    def \_\_init\_\_(self):

        self.head=None        # 头指针

    def len(self):            # 求链表长度

        cur=self.head; length=0

        while cur!=None:

            length=length+1

            cur=cur.next

        return length

    def printf(self):            # 遍历链表

        node=A.head

        for i in range(A.len()):

            print(node.data)

            node=node.next

    def append(self,data):   # 在链表末尾添加节点

        node=Node(data)

        if self.len()==0:

            self.head=node

        else:

            cur=self.head

            pre=None

            while cur!=None:

                pre=cur

                cur=cur.next

            pre.next=node

    def insert(self,sp,data):  # 在指定位置插入新节点

        node=Node(data)

        if sp<=0:

            node.next=self.head

            self.head=node

        elif sp>=self.len():

            self.append(data)

        else:

            cur=self.head

            for i in range(sp-1):

                cur=cur.next

            node.next=cur.next

            cur.next=node

    def pop(self,sp=None):     # 删除指定位置的节点（若sp不填，则默认删除末尾节点）

        if sp==None:

            if self.len()>1:

                cur=self.head

                pre=None

                length=self.len()

                for i in range(length-1):

                    pre=cur

                    cur=cur.next

                pre.next=None

            else:

                self.head=None

        else:

            if self.len()-1<=sp:

                self.pop()

            else:

                cur=self.head

                pre=None

                for i in range(sp):

                    pre=cur

                    cur=cur.next

                pre.next=cur.next

                cur.next=None

    def insert\_1(node,data):   # 在指点节点node后插入一个新节点

        newnode=Node(data)

        node1=node.next

        node.next=newnode

        newnode.next=node1

    def pop\_1(node):          # 删除指定节点node后的那个节点

        node1=node.next

        node2=node1.next

        node.next=node2

        node1.next=None

    def found(self,data): # 判断链表中是否含有数据data的节点，有就返回True

        cur=self.head

        while cur!=None:

            if cur.data==data:

                return True

            cur=cur.next

        return False

A=LinkList()         # 创建链表

for i in range(5):

    A.append(i)

0 1 2 3 4

A.insert(2,7)        # 在第三个节点的位置插入新节点（数据为7）

0 1 7 2 3 4

A.pop(1)             # 删除第二个节点

0 7 2 3 4

A.printf()           # 遍历并输出链表各节点数据

print(A.len())       # 输出链表长度

5

print(A.found(6))    # 判断链表内是否含有某节点数据为6

False

# 基础语句

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | C | JAVA | LUA | PYTHON | PHP | R | JS |
| 注释 |  |  | 行：“- -“（两横杠）  块：”- - [[  ]]” | 行：“#”  块：”””xxx””” | 行：”//”  块：”/\*  \*/” | “#” | 行：”//”  块：”/\*  \*/” |
| 赋值 |  |  | a=1 (可一次性多变量赋值) | a=1 (可一次性多变量赋值) | $a=1 | a<-1 | a=1 |
| 赋值运算缩写 |  |  |  | x+=y: x=x+y  x++: 先赋值后运算  ++x: 先运算后赋值 | x+=y: x=x+y  x++: 先赋值后运算  ++x: 先运算后赋值 |  | x+=y: x=x+y  x++: 先赋值后运算  ++x: 先运算后赋值 |
| 输入 |  |  | x=io.read() | x=input()（默认为字符串，可用int()函数变为数字） |  |  |  |
| 输出 |  |  | print() 默认换行  io.write() 不换行 | print() 默认换行  print(xx,end=” “)不换行 | echo +内容默认输出，不换行);  print() ；sprintf()  向量print\_r() | 内容默认输出 | document.write() 默认不换行 |
| 条件语句 |  |  | If i<=5 then  end  if……then  else  end  if……then  elseif…… then  else  end | If i<=5:  xxx  if……:  xxx  else:  xxx  if……:  xxx  elif……:  xxx  else:  xxx | if(&i<=5){  }  if(……){  }else{  }  if (……){  }else if(……){  }else{  }  switch{  case1……: ;break;  case2……: ;break;  } | if(i<=5){  }  if(……){  }else{  }  if (……){  }else if(……){  }else{  }  switch{  case1……: ;break;  case2……: ;break ;  } | if(i<=5){  }  if(……){  }else{  }  if (……){  }else if(……){  }else{  }  switch{  case1……: ;break;  case2……: ;break ;  } |
| 循环语句 |  |  | 循环句格式：  i=1,100  i=1,100,2  i=100,1,-1  for……do  end  while……do  end  repeat  until…… | i=1  while i<=100:  xxx  i=i+1  for i in range(0,100):  xxx (注意是10而非99)或range(99,-1,-1): | 循环句格式：  $i=1;$i<=100;$i++  $i=0;$i<=100;$i=$i+2  $i=100;$i>=1;$i=$i-1  for(……){  }  while(……){  } | 循环句格式：  i in 1:100  for(……){  }  while(……){  } | 循环句格式：  i=1;i<=100;i++  i=0;i<=100;i=i+2  i=100;i>=1;i=i-1  for(……){  }  while(……){  } |
| 遍历语句（依次逐个遍历所有元素，一般用于字典） |  |  | 遍历句格式：  key, value in pairs(c)  for……do  end | for key,value in A.items():  print(key,value)  或者  for key in A:  print(key,A[key]) | 遍历句格式：  $A as $key=>$value  foreach(……) {  } |  |  |
| 其他 |  |  |  |  |  |  |  |

\*补充：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | C | JAVA | LUA | PYTHON | PHP | R | JS |
| 单行语句结束标志 |  |  | 换行即结束 | 换行即结束 | 分号 ； | 换行即结束 | 换行即结束 |
| 是否严格遵循缩进规则 |  |  | 否 | 是 | 否 | 否 | 否 |

# 基础运算符

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | C | JAVA | LUA | PYTHON | PHP | R | JS |
| 算术运算符 |  |  | +, -, \*, /,^ | +, -, \*, /,\*\* | +, -, \*, /,\*\* | +,-,\*,/,^ | +,-,\*,/,\*\* |
|  |  |  | 模余数: a%b  模商: (a-a%b)/b  取整: Int() | 模余数: a%b  模商: a//b  取整: Int() | 模余数: a%b  模商: (a-a%b)/b  取整: (int) | 模余数: a%%b  模商: (a-a%%b)/b  向上取整:ceiling  向下取整:floor | 模余数: a%b  模商: (a-a%b)/b |
| 关系运算符 |  |  | >,>=,<,<= | >,>=,<,<= | >,>=,<,<= | >,>=,<,<= | >,>=,<,<= |
|  |  |  | ==, ~= | ==,!= | ==,!=  ===,!===（同时判断类型与内容） | ==,!= | ==,!= |
| 逻辑（布尔）运算符 |  |  | and, or, not  x=(x or y)  x为空时取y  x=(A and b) or c  A为假时取c，为真取b | and, or, not | &&, ||, ! | &&, ||, ! | &&, ||, ! |
| 三目运算符  if(b<=c) d=c; else d=b |  |  |  | d = c if b<=c else b | $d=$b<=$c ? $c:$b |  | d=b<=c ? c:b |
| 其他 |  |  | (a or 0)  a为空值时取0，否则取非空值（注意是空值None而不是未定义的值！） | (a or 0)  a为空值时取0，否则取非空值 |  |  |  |

快捷键补充：

Ctrl+Z 取消操作 Tab 多行缩进 Ctrl+Tab 多行退格 Ctrl+/ 多行注释（取消多行注释）

# 基础函数库

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | C | JAVA | LUA | PYTHON | PHP | R | JS |
| 函数 |  |  | f(a,b) | f(a,b) | f(a,b) | f(a,b) | f(a,b) |
| \*函数定义 |  |  | function f(a,b)  end | def f(a,b): | function f(a,b){  } | f(a,b)<-function(){  }  注意函数return()加括号 | function f(a,b){  } |
| 返回值格式 |  |  | return() | return |  |  |  |
| 数学函数格式 |  |  | math.xx() | math.xx() | xx() | xx() | Math.xx() |

\*补充

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | C | C++ | LUA | PYTHON | PHP | R | JS |
| 安装函数包 |  |  |  | Cmd: pip install xx  (删除则为uninstall) |  | 建议用Rstudio下载 |  |
| 调用并查看函数包 |  |  |  | 调用包: import xx  调用包中指定函数：form 包 import 函数  查看包: help(dir(xx)) |  | 建议用Rstudio函数库查看 |  |
| 调用其他文件  （注意主程序和包位于同一父目录下！） |  |  | require(“路径/包”) | Import 路径.包（模块）  或 from 路径.包 import 函数 |  | Library（路径/包） |  |

# 文件（数据集）操作：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | C | JAVA | LUA | PYTHON | PHP | R | JS |
| 读取文件 |  |  | file1=assert(io.open(“xxx”,”r”))  file1:read()  file1:close() | file1= open(“xxx”,”r”,encoding=”utf-8”)  file1.read()  file1.close() | $file1=fopen(“xxx”,”r”);  $a=fgets($file1,size);  （按行读取字符串）  $a=fread($file1,size);（读取定长字符串）  fclose($file); | data1=read.csv(“xxx”) |  |
| 写入文件 |  |  | file1=assert(io.open(“xxx”,”w”))  file1:write()  file1:close() | file1= open(“xxx”,”w”,encoding=”utf-8”)  file1.write()  file1.close() | $file1=fopen(“xxx”,”r”);  fwrite($file1,内容);  fclose($file); | sink("文件名", append=TRUE) |  |
| 其他 |  |  |  |  | $file1=fopen(“xxx”,”a”);用于追加文件内容 |  |  |

# 面向对象编程简介

“面向过程”与“面向对象”是编程的两种最基本思想。

不妨将其看作两名程序员，在解决问题之前，“面向过程”会分析出解决问题的大致思路，并将其拆解为一个个步骤，为实现前后步骤的目的，则只需按步骤之间的逻辑关系依次调用函数即可，因此可以说“面向过程”是解决一般问题的最最基本的编程思想；而“面向对象”则是把问题的主体分解为各个对象（也可以说问题的不同考虑方面），对象不是需要实现的步骤，而是相互联系的事物，它们用于担负起在问题解决过程中自身的某一方面的职责。

关于“面向过程”我们无需多作介绍，下面我们先来引入“面向对象”中最重要的类和对象的概念，在这之后，我们会通过举例子来对比“面向过程”和“面向对象”两种编程思想，从而得出各自的优缺点。

1 类和对象

先定义一个关于狗狗的类，其中包括狗狗的属性，以及一个关于自我介绍的方法或函数：

class Dog:                                 # 类：狗狗

    def \_\_init\_\_(self,name,sex,age):       # 属性（变量）： 名字、性别、年龄

        self.name=name

        self.sex=sex

        self.age=age

    def Introduction(self):                # 方法（函数）：作自我介绍

        print("The dog"+self.name+"is a"+self.sex+"dog,and"+str(self.age)+

        "years old now")

# 每个类均由属性（变量）和方法（函数）两部分组成，其中特殊如方法\_init\_是用来初始化或定义类属性

# 注意，参数self仅用来封装和传递类属性，自身并无任何含义

将类实例化为对象，进一步访问其属性值和调用其方法：

dog1=Dog("Bob","male",3)                    # 类实例化为不同的对象

dog2=Dog("Mary","female",2)

print(dog1.name,dog1.sex,dog1.age)          # 访问不同对象的属性值

print(dog2.name,dog2.sex,dog2.age)

dog1.Introduction()                           # 调用不同对象的方法

dog2.Introduction()

# 类的实例化本质上是调用方法\_init\_并对其中的参数赋值，python中方法\_init\_不作显示

# 由于类已经实例化为对象了，那么当调用某一对象的方法时，参数self（name、sex、age）已知。python中参数self不作显示

Bob male 3

Mary female 2

The dog Bob is a male dog,and 3 years old now

The dog Mary is a female dog,and 2 years old now

允许类中存在公共属性，实例化之后，所有对象公共属性值默认相同。同时允许某一对象修改该属性值

class Dog:

    def \_\_init\_\_(self,name,age):

        self.name=name

        self.sex="male"                         # 公共属性 sex=“male”

        self.age=age

    def Introduction(self):

        print("The dog "+self.name+" is a "+self.sex+" dog,and "+str(self.age)+

        " years old now")

dog1=Dog("Bob",3); print(dog1.sex)

dog2=Dog("Mary",2); print(dog2.sex)    # 类的实例化过程中，公共属性无需赋值（即使赋值为female也不作用）

dog2.sex="female"; print(dog2.sex,dog1.sex)     # 对某一对象修改属性sex的值，但不影响其他对象

male

male

female male

更新某一对象中公共属性值也可采用如下方法：

class Dog:

    def \_\_init\_\_(self,name,age):

        self.name=name

        self.sex="male"

        self.age=age

    def Introduction(self):

        print("The dog "+self.name+" is a "+self.sex+" dog,and "+str(self.age)+

        " years old now")

    def Update\_sex(self,new\_sex):

        self.sex=new\_sex

dog1=Dog("Bob",3); print(dog1.sex)

dog2=Dog("Mary",2); print(dog2.sex)

dog2.Update\_sex("female"); print(dog2.sex,dog1.sex)

male

male

female male

class Dog:

    def \_\_init\_\_(self,name,age,sex="male"):          # 属性sex若未赋值，则实例化后默认为male

        self.name=name

        self.sex=sex

        self.age=age

    def Introduction(self):

        print("The dog "+self.name+" is a "+self.sex+" dog,and "+str(self.age)+

        " years old now")

dog1=Dog("Bob",3); print(dog1.sex)

dog2=Dog("Mary",2); print(dog2.sex)

dog2=Dog("Mary",2,"female"); print(dog2.sex,dog1.sex)

male

male

female male

最后我们简单来介绍类的继承和调用这一概念：

class Dog:

    def \_\_init\_\_(self,name,age):

        self.name=name

        self.sex=None

        self.age=age

    def Introduction(self):

        print("The dog "+self.name+" is a "+self.sex+" dog,and "+str(self.age)+

        " years old now")

class Movement:

    def \_\_init\_\_(self,sex,movement="running"):

        self.sex=sex

        self.movement=movement

    def describe(self):

        if self.sex=="male":

            print("Look, he is "+self.movement)

        else:

            print("Look, she is "+self.movement)

class Male\_Dog(Dog):                             # 子类Male\_Dog继承父类Dog中所有方法(\_init\_和Introduction)

    def \_\_init\_\_(self, name, age,action):

        super().\_\_init\_\_(name, age)              # 方法super().\_\_init\_\_使子类继承父类中所有非公共属性

        self.sex="male"                          # 在子类中增加属性sex="male"

        self.action=action

        self.move=Movement(self.sex,action)       # 将类Movement实例化，并作为Male\_Dog的一个属性来调用

class Female\_Dog(Dog):                             # 同理

    def \_\_init\_\_(self, name, age,action):

        super().\_\_init\_\_(name, age)

        self.sex="female"

        self.action=action

        self.move=Movement(self.sex,action)

dog=Male\_Dog("Tom",4,"sitting")

dog.Introduction()

dog.move.describe()

dog=Female\_Dog("Judy",3,"sleeping")

dog.Introduction()

dog.move.describe()

The dog Tom is a male dog,and 4 years old now

Look, he is sitting

The dog Judy is a female dog,and 3 years old now

Look, she is sleeping

2 面向过程与面向对象编程的特点：

例子：求e的1000位小数并打印出来

def concat(Table,s1,start,end):                                # 列表元素拼接

    s=str(Table[0])

    for i in range(start+1,end+1):

        s=s+s1+str(Table[i])

    return s

#面向过程编程

M=[1]; N=[1]; sp=0; dividend=1

while sp<=999:                                                 # 若小数位小于1000位，则继续运算；反之停止运算

    for j in range(sp,1010):

        if len(N)-1>=j+1: N[j+1]=N[j+1]+(N[j]%dividend)\*10     # 等价于N[j+1]=(N[j+1] or 0)+(N[j]%dividend)\*10

        else: N.append((N[j]%dividend)\*10)

        N[j]=int((N[j]-N[j]%dividend)/dividend)

        if len(M)-1>=j: M[j]=M[j]+N[j]

        else: M.append(N[j])

    while N[sp]==0:

        sp=sp+1

    dividend=dividend+1

for j in range(1009,-1,-1):                                   # 十进制调整

    M[j-1]=M[j-1]+int((M[j]-M[j]%10)/10)

    M[j]=M[j]%10

print(concat(M,"",0,999))

优点：遵循自顶向下的串联式逻辑，程序简洁，效率高，对于复杂问题的处理有一定优势

缺点：对于类似问题较难直接推广，有时需要作较大改动

def concat(Table,s1,start,end):                            # 列表元素拼接

    s=str(Table[0])

    for i in range(start+1,end+1):

        s=s+s1+str(Table[i])

    return s

#面向对象编程

class Division:                                            # 数组除法运算

    def \_\_init\_\_(self,Arr,dividend=1):

        self.Arr=Arr

        self.dividend=dividend

    def division(self):

        for j in range(len(self.Arr)-1):

            self.Arr[j+1]=self.Arr[j+1]+(self.Arr[j]%self.dividend)\*10

            self.Arr[j]=int((self.Arr[j]-self.Arr[j]%self.dividend)/self.dividend)

        return self.Arr

class Product:                                            # 数组乘法运算

    def \_\_init\_\_(self,Arr,exponent):

        self.Arr=Arr

        self.exponent=exponent

    def product(self):

        for j in range(len(self.Arr)):

            self.Arr[j]=self.Arr[j]\*self.exponent

        for j in range(len(self.Arr)-1,-1,-1):

            self.Arr[j-1]=self.Arr[j-1]+int((self.Arr[j]-self.Arr[j]%10)/10)

            self.Arr[j]=self.Arr[j]%10

        return self.Arr

class Sum:                                              # 数组之间加和运算

    def \_\_init\_\_(self,Arr,Arr1):

        self.Arr=Arr

        self.Arr1=Arr1

    def sum(self):

        for j in range(len(self.Arr1)):

            self.Arr1[j]=self.Arr[j]+self.Arr1[j]

        for j in range(len(self.Arr1)-1,-1,-1):

            self.Arr1[j-1]=self.Arr1[j-1]+int((self.Arr1[j]-self.Arr1[j]%10)/10)

            self.Arr1[j]=self.Arr1[j]%10

        return self.Arr1

class Count:                                            # 找出数组元素的非零起始位置

    def \_\_init\_\_(self, Arr):

        self.Arr=Arr

        self.sp=0

    def update\_sp(self):

        while self.Arr[self.sp]==0 and self.sp<1009:

            self.sp=self.sp+1

        return self.sp

class Compute:                                          # 计算e的任一指数的1000位小数

    def \_\_init\_\_(self,Arr,Arr1,exponent):

        self.Arr=Arr

        self.Arr1=Arr1

        self.exponent=exponent

        self.dividend=1

        self.sp=Count(self.Arr).update\_sp()

    def compute(self):

        while self.sp<=999:

            self.Arr=Product(self.Arr, self.exponent).product()[:]

            self.Arr=Division(self.Arr, self.dividend).division()[:]

            self.Arr1=Sum(self.Arr, self.Arr1).sum()[:]

            self.sp=Count(self.Arr).update\_sp()

            self.dividend=self.dividend+1

        print(concat(self.Arr1,"",0,999))

        return self.sp

Arr=[1]; Arr1=[1]; exponent=1

for j in range(1,1010): Arr.append(0); Arr1.append(0)

A=Compute(Arr,Arr1,exponent)

A.compute()

优点：遵循分布式逻辑，可扩展性较强，处理类似问题仅需要对部分程序做修改

缺点：程序较为复杂，对于某些复杂问题不易处理

# 递归编程及其他编程思想简介

递归编程基本思想：

在面向对象编程中，我们学会了将待解决的问题主体分为各个对象，每个对象担负起部分功能，然后根据各个对象之间的联系来相互调用，并最终解决问题。现在，我们要学习一种新的编程思想—递归编程。通过定义递归函数，递归编程可以将最初的大问题拆解为许许多多性质相同的中间问题，所有中间问题的关系类似于“树”的结构，把大问题比作“树顶”，大问题逐渐向下拆解，在抵达“树底”之后，再逐步递归，返回“树顶”。

递归编程基本条件：

（1）必须有递归终止条件，不然会永远执行下去！

（2）必须使得某一状态逐渐接近终止条件，类比于由“树顶”向“树底”靠拢。

（3）在执行递归函数并使上述状态接近终止条件过程中，必须不断调用函数自身

我们举个简单的例子来展示递归编程的思想：

# 用“递归”代替“循环”计算 1+2+3+……+9

# 循环方法

sum=0

for i in range(1,10):

sum=sum+i

# 递归方法：通过执行sum(n-1)使得n不断减小靠近1，在n=1之后，开始朝着n增大的方向返回 sum(n-1)+n的值：1、1+2、1+2+3………

def sum(n):

    if n<=1: # 满足终止条件，返回sum(1) = 1的值至sum(2)

        return 1

return sum(n-1)+n # 满足终止条件前，一直执行sum(n-1)；在满足条件后，不断返回sum(n) = sum(n-1) + n的值至上一层

n:  10 -> 9 -> 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1（抵达终止条件）

   (1+2+3+……+10) <- …… <- (1+2+3) <- (1+2) <- (1) :sum （开始返回sum(n-1)+n的值）

可以看到相比于一般方法，递归编程并未使用循环语句，但同样计算出相同的结果！在后续的应用中，我们会看到，递归编程不仅能够化简代码，往往还能解决常规编程方法所解决不了的问题！

# （1）十进制换二进制

def Ten\_to\_Two(n):

    if n<=1:

        print(int(n),end="")

    else:

        Ten\_to\_Two((n-n%2)/2)

        print(int(n%2),end="")

Ten\_to\_Two(18)

10010

# （2）汉诺塔问题:共有n个盘子，求把所有盘子正确地从柱子A移到柱子C的操作？

# 思路：先把前n-1个盘子正确移到柱子B，然后把第n个盘子移到柱子C，再把前n-1个盘子从柱子B移到柱子C

def MoveTower(n,start="A",middle="B",end="C"):

    if n==1:

        print(start+"->"+end)  # 终止条件：只有一个盘子时，停止调用函数，直接将盘子移到目标柱子

    else:

        MoveTower(n-1,start,end,middle)

        MoveTower(1,start,middle,end)

        MoveTower(n-1,middle,start,end)

MoveTower(5)

A->C

A->B

C->B

A->C

B->A

B->C

A->C

A->B

C->B

C->A

B->A

C->B

A->C

A->B

C->B

A->C

B->A

B->C

A->C

B->A

C->B

C->A

B->A

B->C

A->C

A->B

C->B

A->C

B->A

B->C0

A->C

最后我们简单介绍一下其他常见的编程思想：

（1）贪婪算法：贪婪算法旨在获取小问题的局部最优解，进而获取大问题的全局最优解或近似全局最优解，但获取最优解的优劣要看贪婪策略的选择，如下：

#背包问题：

"""

一小偷携带负荷20的背包来珠宝店偷东西，现列出物品重量和价值以供考虑:

(1,3),(2,4),(3,5),(5,6),(6,8),(8,10),(12,15),(13,20)，问小偷采取如下何种贪婪策略，可偷取得最大价值:

【1】在不超过负荷情况下，每次价值最大【2】在不超过负荷情况下，每次平均价值最大

"""

max=20

Goods\_Value=[[1,3],[2,4],[3,5],[5,6],[6,8],[8,10],[12,15],[13,20]]

def f1(max, Goods\_Value):  # 策略1：每次偷取价值最大

    sum\_value=0; sum\_goods=0

    for i in range(len(Goods\_Value)-1,-1,-1):

        sum\_value=sum\_value+Goods\_Value[i][1]

        sum\_goods=sum\_goods+Goods\_Value[i][0]

        if sum\_goods>max:

            sum\_value=sum\_value-Goods\_Value[i][1]

            sum\_goods=sum\_goods-Goods\_Value[i][0]

        else:

            print("("+str(Goods\_Value[i][0])+","+str(Goods\_Value[i][1])+")")

    return sum\_goods,sum\_value

def f2(max, Goods\_Value):  # 策略2：每次偷取平均价值最大

    sum\_value=0; sum\_goods=0

    for i in range(len(Goods\_Value)):

        Goods\_Value[i].append(Goods\_Value[i][1]/Goods\_Value[i][0])

    Goods\_Value.sort(key=(lambda x:x[2]))  # 对二维列表按照关键字x[2]倒序排序

    for i in range(len(Goods\_Value)-1,-1,-1):

        sum\_value=sum\_value+Goods\_Value[i][1]

        sum\_goods=sum\_goods+Goods\_Value[i][0]

        if sum\_goods>max:

            sum\_value=sum\_value-Goods\_Value[i][1]

            sum\_goods=sum\_goods-Goods\_Value[i][0]

        else:

            print("("+str(Goods\_Value[i][0])+","+str(Goods\_Value[i][1])+")")

    return sum\_goods,sum\_value

print(f1(max,Goods\_Value))

(13,20)

(6,8)

(1,3)

(sum\_goods=20, sum\_value=31)

print(f2(max,Goods\_Value))

(1,3)

(2,4)

(3,5)

(13,20)

(sum\_goods=19, sum\_value=32)

（2）动态规划算法：相比于贪婪算法中将大问题分为小问题，并求每个小问题的最优解的方法，动态规划算法将某个小问题逐步扩展为大问题，并在这一过程始终求其最优解，那么可想而知，最后求得的最优解一定是全局最优解！

"""

同样是背包问题，在上题中我们还可以用动态规划算法来求解，定义动态规划矩阵，横行代表所考虑的物品个数，从物品(1,3)直到全部物品

；纵列代表背包负荷的变化，从负荷为1到负荷为20。以i行j列为例，此时我们需要的问题是:在考虑前i个物品且负荷为j的情况下，可偷取

的最大价值。不难得知当G[i][0]>=j时，A[i][j]=max(A[i-1][j], G[i][1]+A[i-1][j-G[i][0]])

"""

max=20

Goods\_Value=[[1,3],[2,4],[3,5],[5,6],[6,8],[8,10],[12,15],[13,20]]

def f(max,Goods\_Value):

    A=[]; length=len(Goods\_Value)

    for i in range(length+1):

        A.append([])

        for j in range(max+1):

            A[i].append(0)

    for i in range(1,length+1):

        for j in range(1,max+1):

            A[i][j]=A[i-1][j]

            if Goods\_Value[i-1][0]<=j and A[i-1][j]<Goods\_Value[i-1][1]+A[i-1][j-Goods\_Value[i-1][0]]:

                A[i][j]=Goods\_Value[i-1][1]+A[i-1][j-Goods\_Value[i-1][0]]

    for i in range(length,-1,-1): # 根据动态规划矩阵A的结果，我们从A[length][max]一直回溯至A[1][1]，判断G[i-1]是否被偷取

        if A[i][max]==Goods\_Value[i-1][1]+A[i-1][j-Goods\_Value[i-1][0]]:

            max=max-Goods\_Value[i-1][0]

            print(Goods\_Value[i-1])

f(max,Goods\_Value)

[13, 20]

[3, 5]

[2, 4]

[1, 3]

（3）回溯算法：在处理非线性的树和图问题时，我们往往需要沿着一条路径向下走，如果能走通，则问题解决，反之需要回到上一步并继续尝试，通过大量的穷举试验，我们最终总能找到解决问题的一条路径。回溯算法的应用与递归编程思想有十分紧密的关系，如下：

# 马的遍历问题:nxm的棋盘，选定棋盘上一点(x,y)并按照“马”的规则走，规划出该点不重复地走过棋盘上任一位置的路线？

class Check: # 用于判断走位的正确性

    def \_\_init\_\_(self,n,m,x,y):

        self.n=n

        self.m=m

        self.x=x

        self.y=y

        self.A=[]

        for i in range(self.n):

            self.A.append([])

            for j in range(self.m):

                self.A[i].append(0)

    def check(self):

        if (self.x<1 or self.x>self.m) or (self.y<1 or self.y>self.n) or self.A[self.x-1][self.y-1]!=0:

            return False

        return True

class Run(Check): # 用于递归走位

    def \_\_init\_\_(self, n, m, x, y):

        super().\_\_init\_\_(n, m, x, y)

        self.step=None

        self.D\_x=[-2,-2,2,2,1,-1,1,-1]

        self.D\_y=[1,-1,1,-1,-2,-2,2,2]

    def run(self):

        for i in range(8):

            if self.check():

                self.A[self.x-1][self.y-1]=self.step

                if self.step==self.n\*self.m:

                    return self.A

                else:

                    self.step=self.step+1

                    self.run()

                    self.step=self.step-1

                    self.A[self.x-1][self.y-1]=0

            else:

                self.x=self.x-self.D\_x[i]; self.y=self.y-self.D\_y[i]

class Game(Run): # 用于初始化棋盘和起始点，并输出路线

    def \_\_init\_\_(self, n, m, x, y):

        super().\_\_init\_\_(n, m, x, y)

        self.A[self.x-1][self.y-1]=1

        self.step=2

    def game(self):

        if self.run()!=None:

            for i in range(len(self.A)):

                print(concat(self.A[i],"\t"))

        else:

            print("try again!")

p=Game(5,5,1,1)

p.run()

1       14      3       8       21

4       9       20      13      16

19      2       15      22      7

10      5       24      17      12

25      18      11      6       23

为了避免重复遍历，通过某种“剪枝方法”，回溯算法可达到以空间换时间，从而实现优化，此处不做过多介绍。

# 程序异常与调试

Python中程序异常可分为两种，第一种是语法错误（通常编辑器会提示），第二种是逻辑错误（需要自己调试），如下：

#语法错误

print 3

File "c:\Users\熊高庆\Desktop\项目文件\Python\Python程序\Learning\Datebase Learning\training.py", line 1

    print 3

          ^

SyntaxError: Missing parentheses in call to 'print'. Did you mean print(3)?

File指代程序异常的文件

line1表示错误出现在第一行，并给该行代码

SyntaxError表示出现语法错误，错误信息Missing parentheses表示缺少圆括号

Did you mean print(3)? 给出了修改建议

#逻辑错误

print(5/0)

Traceback (most recent call last):

  File "c:\Users\熊高庆\Desktop\项目文件\Python\Python程序\Learning\Datebase Learning\training.py", line 3, in <module>

    print(5/0)

ZeroDivisionError: division by zero

Traceback指异常信息追溯

File 和 line同理

ZeroDivisionError：division by zero 表示犯了“零除”错误

在一个程序中，对于可能出现异常的代码，我们可用try……except语句来处理，如下：

#处理前

x=int(input()); y=int(input())

print(x/y)

print(x\*y)

4

0

Traceback (most recent call last):

  File "c:\Users\熊高庆\Desktop\项目文件\Python\Python程序\Learning\Datebase Learning\tempCodeRunnerFile.py", line 2, in <module>

    print(x/y)

ZeroDivisionError: division by zero

当y取0时，x/y则犯了零除错误，从而给出异常信息并终止程序

#处理后

x=int(input()); y=int(input())

try:

    print(x/y)

except ZeroDivisionError:                 # ZeroDivisionError也可换成一般的Exception

    print("divide by 0")

    print(0)

print(x\*y)

4

0

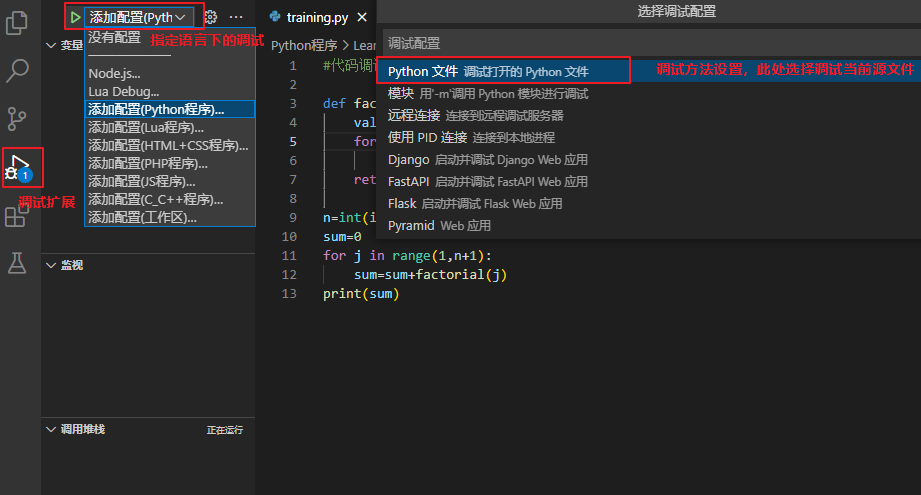
divide by 0

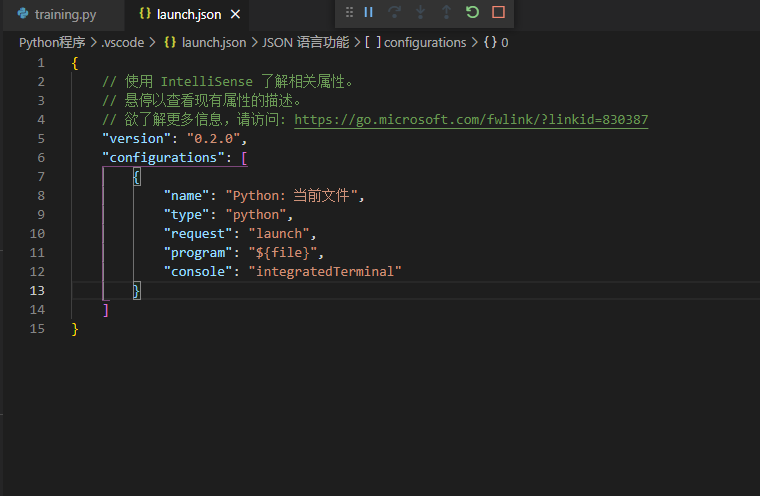
0

0

我们将可能出现异常的代码块用 try语句包含，然后用except语句给出异常信息提示，以及异常时的默认值0，这样做的好处是不会因为出现异常就终止程序，后面的print(x\*y)同样可以执行。关于具体的异常信息以及try……except语句的更多用法，我们不作更多介绍。

当出现程序异常时，最好的处理方法当然是调试代码，并将异常修正。下面我们给出Vscode中代码调试的方法：





此处会生成一个关于调试信息的文件，其中：

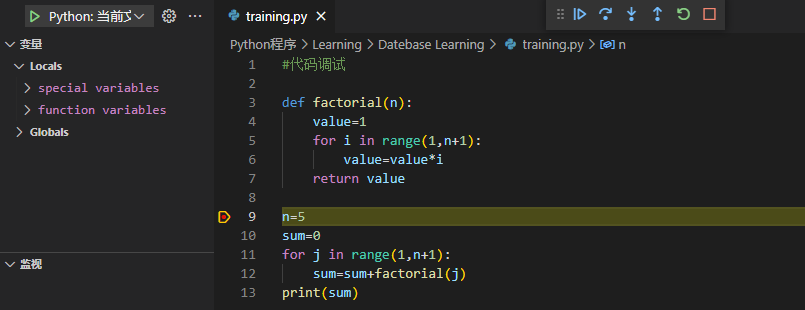
Name：表示当前调试配置的名称

Type：表示对哪种语言进行调试

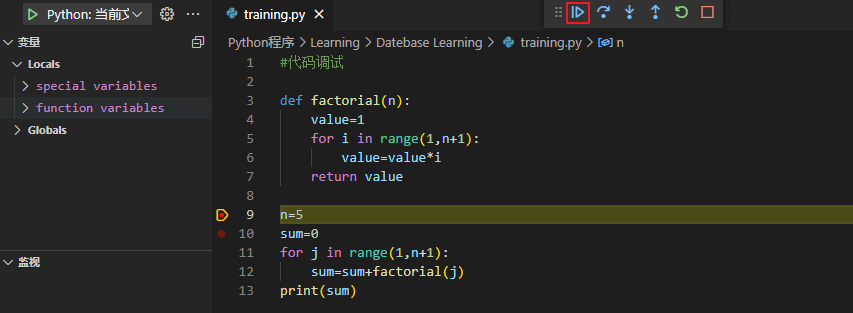
Request：调试方式，通常为launch

Program：调试文件的绝对路径

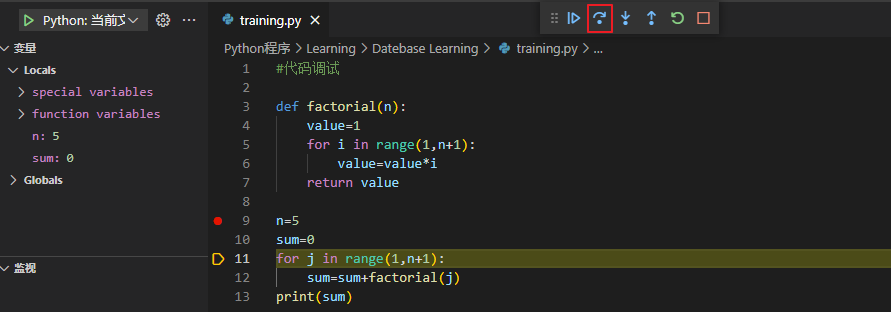
Console：终端类型，通常为IntegratedTerminal

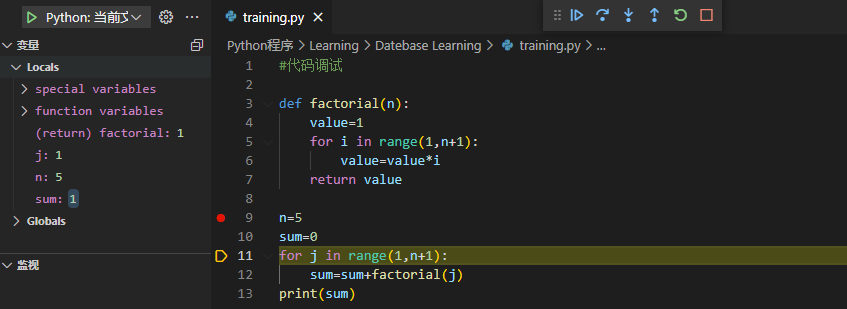


在第九行设置一个断点，则进入调试后，代码会直接运行到断点的前一行（第八行），此时变量n的信息在左侧还未显示

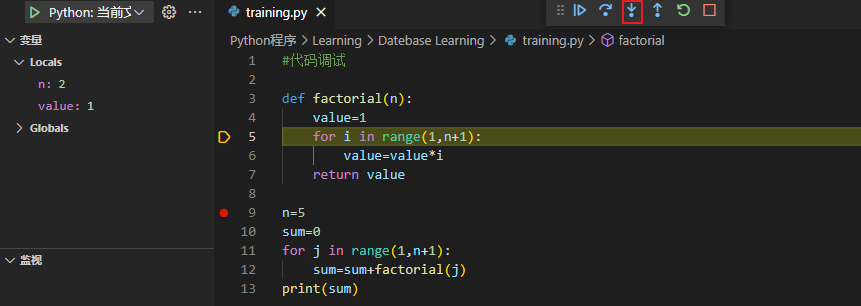


若点击“继续”，则直接运行断点及以后的全部程序，调试结束并在终端打印出sum的结果，此时左侧并不显示调试过程中变量值的信息，因此在调试中一般不直接点击“继续”。

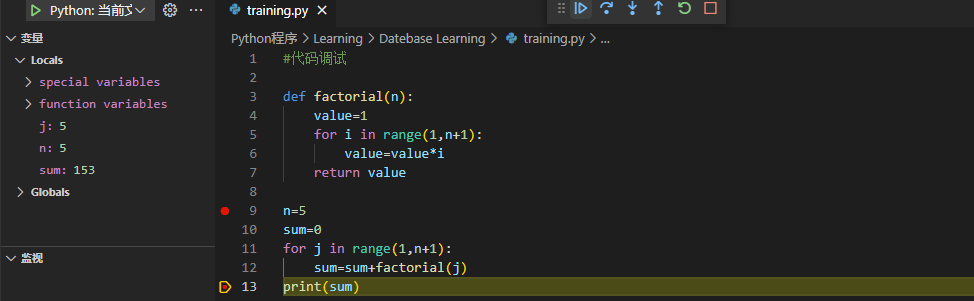




若点击“单步跳过”，则当程序运行至第十一行时（实际运行了前十行），变量n和sum的信息给出。进一步执行进入循环，除变量n和sum外，左侧还给出了变量j以及j的函数factorial的信息。



若点击“单步调试”，则程序会由执行逻辑依次运行，此时调试可进入函数factorial内部，并在左侧给出相应变量的信息。



除此之外，还可以设置多个断点来进行调试！

# 算法分析

算法是用于解决问题的一种方法或思路，而程序是算法的具体化，不仅包括了算法，还关系到用哪种语言，程序如何设计，甚至是用哪种编译器，哪台计算机等等。因此一个程序的好坏不仅与算法有关，还涉及到其他各各方面。

在实际应用中，我们更关心程序的好坏评价，一般是看它的实际运行时间和所占用的内存，CPU等等，如下

import time

start = time.time()    # 查看当前时间，以秒为单位

#---------------------------

def factorial(n):

    value=1

    for i in range(1,n+1):

        value=value\*i

    return value

n=5

sum=0

for j in range(1,n+1):

    sum=sum+factorial(j)

print(sum)

#--------------------------

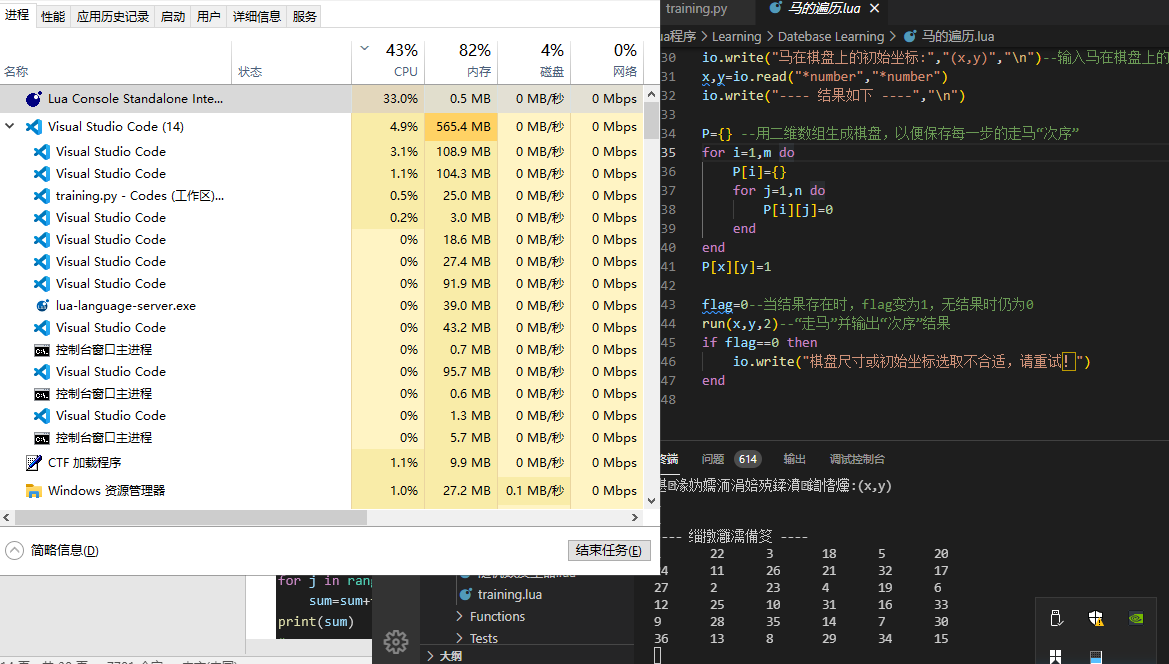
end = time.time()

print(end-start)      # 输出起止时间之差

153

0.0009839534759521484

可见程序运行时间大约为0.00098秒，但需要注意这不一定是程序运行的真实时间，有可能会受到当前正在运行的其他应用的影响。

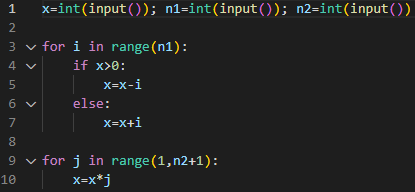


针对某些运行时长较长的程序，可用ctrl+shift+esc打开任务资源管理器，动态监控在当前计算机下，程序运行在当前计算机下所占内存和CPU。

而在算法研究中，排除其他外在因素，为了对比程序运行前算法的优劣，我们需要关注两个指标，时间复杂度和空间复杂度。

1. 时间复杂度：

对于某个程序，根据它每一行代码是否涉及到cpu运算，我们可提取出所有的有效代码行。然后再计算每一行的执行次数（考虑到循环，条件语句等特殊结构），最终加和可得该程序的总执行次数T(n)，示例如下：



可知其有效代码行为3，5，7，9，10，则总执行次数为n1 + n1 + n2 + n2 = 2n1 + 2n2（在条件语句中，5和7只能选其中一条执行）。

考虑n1,n2趋于无穷，则总执行次数T(n)为4n(n趋于无穷)，可知T(n)是关于n的正比函数，即总执行次数T(n)随着n单位增长（以4个单位），由此我们定义该程序的时间复杂度为O(n)

下面我们列出几种常见的时间复杂度O()：

# 常数阶

x=1

print(x+1)

print(x-1)

# T(n)=2, O=O(1)

# 一次线性阶

x=1

for i in range(n):

    x=x+i

    x=x\*i

print(x)

# T(n)=2n, O=O(n)

# 对数阶

x=1; i=1

while i<=n:

    x=x+i

    i=i\*2

print(x)

# T(n)=2log2(n), O=O(log2(n))

# 二次线性阶

x=1

for i in range(n):

    for j in range(n):

        x=x+i

        x=x\*j

print(x)

# T(n)=2n^2, O=O(n^2)

# 二次线性阶（特殊）

x=1

for i in range(n):

    for j in range(i):

        x=x+i

        x=x\*j

print(x)

# T(n)=1+2+3+……+n = (n^2)/2+n/2 （考虑n为偶数）

# O=O(n^2) (考虑n的最高次项)

已知，各种时间复杂度之间的关系为O(1)<O(log(n))<O(n)<O(n^2)

1. 空间复杂度：

我们定义某个程序在运行过程中为辅助变量（自定义变量）所分配的最大储存空间的函数为空间复杂度O（类比于时间复杂度），举例如下：

# 单变量

m=1

# O=O(1)

# 向量（数组、列表、字典）

M=[]

for i in range(n):

    M.append(i)

# O=O(n)

# 二维向量 （数组、列表、字典）

M=[]

for i in range(n):

    M[i]=[]

    for j in range(n):

        M[i].append(i\*j)

# O=O(n^2)

特殊地，当程序中含有递归结构时，我们规定如下：

#单变量递归

def f(n):

    if n<=1:

        return 1

    return f(n-1)

print(f(n))

# 总空间复杂度O = O(递归层数)= O(n)

#一维向量递归

N=[]

def f(n):

    if len(N)>=n:

        return N

    N.append(1)

    return f(n)

print(f(n))

# 总空间复杂度O = O(递归层数）= O(最大向量长度) = O(n)

#二维向量递归

N=[]

def f(n):

    if len(N)>=n:

        return N

    N.append([1,2,……,m])

    return f(n)

print(f(n))

# 总空间复杂度O = O(各层空间复杂度之和）= O(列维数\*最大行维数) = O(mn)

以时间复杂度为例，不难看出，上述例子中其时间复杂度大小总是固定的，并不存在最好与最坏的情况，然而更多的，当我们处理某一数据结构时，往往需要考虑其最佳情况，最糟糕情况和平均情况，如下：

# 最佳，最糟糕和平均情况

N=[x1,x2,x3,……,xn]

for i in range(len(N)):

    if N[i]==y:              # 考虑N中有且仅有一个等于y的元素

        break

# 最佳情况 O = O(1) 数组第一个元素即为y

# 最糟糕情况 O = O(n) 数组最后一个元素为y

# 平均情况 O = O( 1/n + 2/n + 3/n + …… + 1) = O(n/2 + 1/2) = O(n) 考虑第j个元素为y，以及该事件的概率，从而求O的期望值

当然多数情况下，我们首先考虑最糟糕的情况，只有当该三种同情况下时间复杂度存在数量级差别时，才可能首要考虑平均情况下的时间复杂度。

最后，当给定不同类型数据结构时，若用同一算法处理该数据结构，其时间复杂度也有可能不同，如下：

# 列表和字典性能比较：判断其是否含有某一元素

#列表

List=[x1,x2,x3,……,xn]

for i in range(len(List)):

    if List[i]==y:

        print(1)

# O=O(n)

Dict={"x1":1,"x2":1,……,"xn":1}

print(Dict["y"])

# O=O(1)

\*补充：python中列表，链表和字典常见操作的时间复杂度

（1）列表：

index[]            O(1)

in                            O(n)

append                  O(1)

insert(i,item)          O(n)

pop                        O(1)

pop(i)   / del(i)         O(n)

sort                        O(nlogn)

reverse(逆序)         O(n)

get slice[x:y]           O(k)(k是X到Y的长度)

copy              O(n)

（2）链表：

index[]            O(n)

in                            O(n)

append                  O(n)

insert(i,data)          O(n) \*头部插入，删除时间复杂度为O(1)

insert\_1(node,data) O(1)

pop                        O(n)

pop(i)        O(n)

pop\_1(node) O(1)

（3）字典

index[]        O(1)

in                   O(1)

set k-v          O(1)

del k-v     O(1)

copy              O(n)

# 数据可视化

1. Python
2. R
3. Echarts

# Python数据爬虫

要想真正熟练并精通爬虫技术并不容易，本文只简单介绍以python为语言基础的常用的爬虫手段，如下：

我们先介绍一种最基础的基于静态网页html源码爬取的技术，来帮助我们了解爬虫的底层原理

# 爬虫基础-爬取文字

import requests    # 导入爬虫模块

from bs4 import BeautifulSoup

"""

假定我们的目标是爬取小白盘网站首页的推荐书籍书名，并将其导入至EXCEL中

目标网址:https://www.xiaobaipan.com/

查看robot协议: 点击https://www.xiaobaipan.com/robots.txt可访问网址源文件根目录中robots声明，一般不允许商业爬虫，个人

用于学习研究影响不大。

打开网址html源码:点击f12 -> elements或元素，利用左上角导航箭头，可用于定位

"""

res = requests.get("https://www.xiaobaipan.com")  # 以国家统计局2021国内生产总值为例，向服务器发出请求并捕捉响应

# print(res.status\_code)  返回结果为200表示请求成功

# print(res.text)  返回html标记语言

html=BeautifulSoup(res.text,"html.parser") # 解析html标记语言

# print(html) 内容与上一条输出信息相同，只是变成了解析后的html

"""

对于最常见的爬虫技术来说，其基本原理十分简单，就是从解析后的html中寻找，定位，抓取我们想要的内容，以文本内容为例，

鉴于html语言的分层特点，可以利用find()【寻找满足条件的第一个】和find\_all【寻找满足条件的全部】层层定位，或者快速

定位文本内容所在位置，其中两个函数的参数均为"("标签","对应标签的属性")

"""

tag1=html.find("div",class\_="main-container")

tag2=tag1.find\_all("h6")

file= open("book\_name.txt","w",encoding="utf-8")  # 将爬取的内容写入txt文件

for tag in tag2:

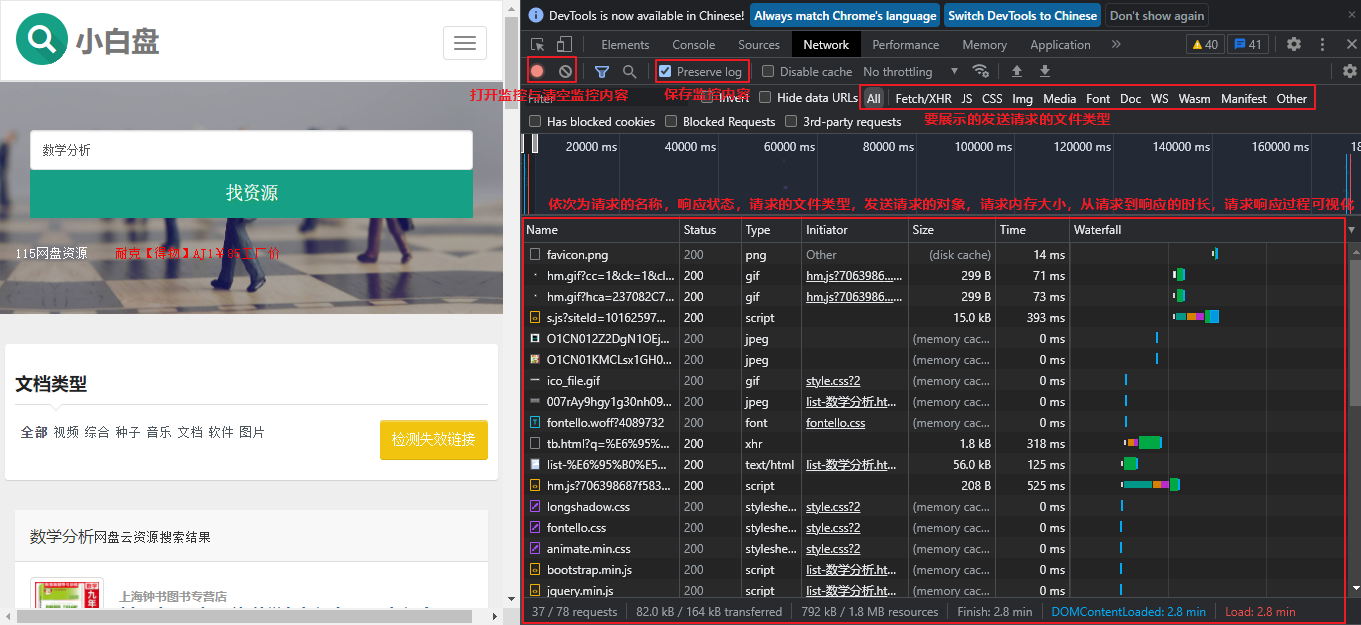
    tag\_root=tag.find("a")  # tag\_root指不可再展开的html标签，即根标签

    # file.write(tag\_root["title"][:-5]+"\n") # tag\_root["title"]用来提取标签的"title"属性值

    file.write(tag\_root.text.strip()+"\n") # tag\_root.text用来提取标签内的text文本内容，效果与上述方法一致

file.close()

借用以上的爬虫方法，一般来说我们可以爬取诸如小说网上的小说，博客网上的博客及评论等等，然而，在实际应用中，我们需要爬取的内容并不来自html源码文件，而是该网站在服务器的文件，如图片，视频，pdf等等，因此显而易见的是，我们并不能从html源码文件中爬取任何关于非文本内容的资源，为什么呢?





同样以小白盘为例，在我们搜索“数学分析”之后，会出现一系列关乎“数学分析”的资源，需注意的是，其中包含了大量的封面图片及头像图片，然而在定位html源码时，我们只能找到打印图片的html指令，其图片由于储存在服务器文件夹中而以链接的形式被调用。因此我们的目的是从服务器文件夹中爬取图片，视频等资源！

上面显示了该页面各种文件（html源码文件一般在DOC中）的请求及响应动态，现在我们来尝试爬取该页面的全部图片

# 爬取图片

import requests    # 导入爬虫模块

import os  # 导入OS文件处理模块

from bs4 import BeautifulSoup

path="C:/Users/熊高庆/Desktop/项目文件/Python/Python程序/Learning/Functions/images"  # 判断该路径下文件夹images是否存在

if not os.path.exists(path):  # 不存在则创建一个新文件images

    os.mkdir(path)

res=requests.get("https://www.xiaobaipan.com/list-SPSS.html?from=1")

html=BeautifulSoup(res.text,"html.parser")

tags=html.find\_all("img")  # 直接暴力搜索标签"img"

Url=[]

for tag in tags:

    Url.append(tag["src"])  # 以上操作类比文字爬虫，仅仅从爬取文字转而为url链接,然后将图片URL链接保存在列表Url中

for i in range(len(Url)):

    try:   # try……except语句用来避免程序因某个子操作异常而终止

        res=requests.get(Url[i])  # 针对图片网络链接发出新的请求

        new\_url=path+"/"+Url[i].split("/")[-1]  # 将网络链接改为本地链接

        file=open(new\_url,"wb")  # 创建本地链接，用二进制形式保存图片信息

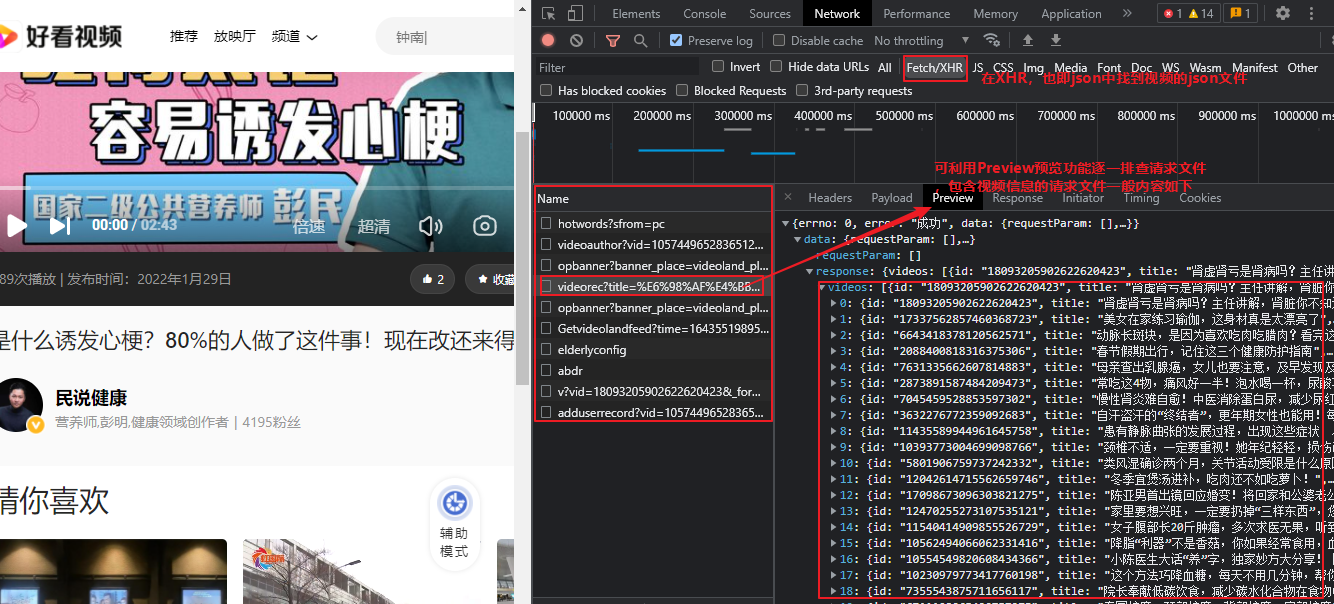
        file.write(res.content)

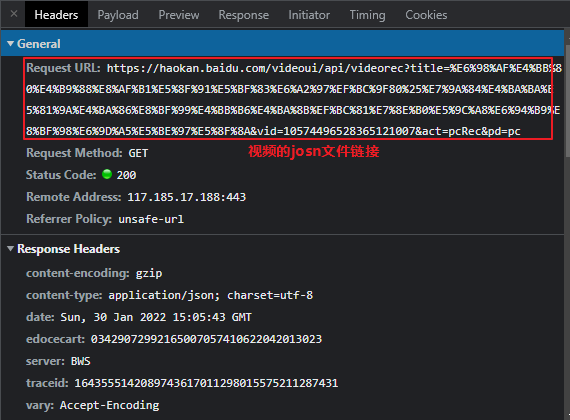
        print("爬取成功")

    except Exception:

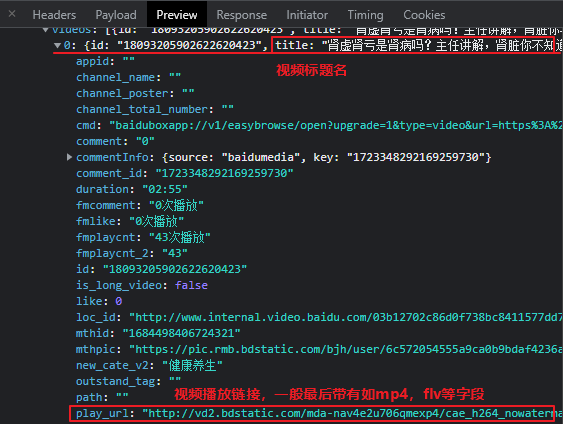
        print("爬取失败")

同样的，我们还可以爬取视频网站的视频，但通常情况下我们会用到json解析模块这一工具，我们以好看视频网站为例，具体如下：





点击XHR，右侧可弹出一个关于该请求的信息框，我们重点关注Headers和Preview两个。利用Preview找到我们需要的那个请求，然后在Headers的Request URL中找到视频的json文件链接，链接可用ctrl+C复制。



在解析完json文件后，我们会得到一个列表和字典交互的数据结构，我们找到最底层的那个列表，里面包含了该页面所有视频的信息，从中找到每一个视频的网络链接，然后依照图片那样即可完成爬虫，完整代码如下：

# 爬取视频

"""

随着网页制作中Ajax技术的发展，制作者往往会利用json文件来封装部分html源码以加快网页的刷新，因此我们直接利用requests获取

的html源码并不完整，自然也就无法找到对应资源（如文字，图片，视频）的内容或链接。一般来说，有两种常用的方法来解决这一问题:

（1）模拟Chrome浏览器来访问网页。（2）将json文件中解析为字典，并从中找到资源内容或链接。本文主要介绍第二种

"""

import requests # 导入爬虫模块

import os  # 导入OS文件处理模块

import json # 导入json模块

from bs4 import BeautifulSoup

path="C:/Users/熊高庆/Desktop/项目文件/Python/Python程序/Learning/Functions/videos"  # 判断该路径下文件夹videos是否存在

if not os.path.exists(path):  # 不存在则创建一个新文件videos

    os.mkdir(path)

Request\_URL="https://haokan.baidu.com/videoui/api/videorec?title=%E6%98%AF%E4%BB%80%E4%B9%88%E8%AF%B1%E5%8F%91%E5%BF%83%E6%A2%97%EF%BC%9F80%25%E7%9A%84%E4%BA%BA%E5%81%9A%E4%BA%86%E8%BF%99%E4%BB%B6%E4%BA%8B%EF%BC%81%E7%8E%B0%E5%9C%A8%E6%94%B9%E8%BF%98%E6%9D%A5%E5%BE%97%E5%8F%8A&vid=10574496528365121007&act=pcRec&pd=pc"

res=requests.get(Request\_URL) # 找到XHR中视频的json链接（可以逐个preview预览排查）

# print(res.text) 打印出来，我们会发现这是一个json文件

json=json.loads(res.text) # 将上述json文件解析为字典

video\_list=json["data"]["response"]["videos"] # 根据preview预览的字典结构进行拆分，找到最底层视频所在的列表

for i in range(5):

    # print(video\_list[i],"\n") 打印列表内各个元素，我们会看到每一个视频的信息，它们是以字典的形式存在

    video\_url=video\_list[i]["play\_url"]  # 找到视频的网络链接

    video\_name=video\_list[i]["title"]+".mp4"  # 将视频重命名

    try:

        res=requests.get(video\_url)

        new\_url=path+"/"+video\_name # 将网络链接改为本地链接

        file=open(new\_url,"wb")

        file.write(res.content) # 用二进制形式保存视频信息

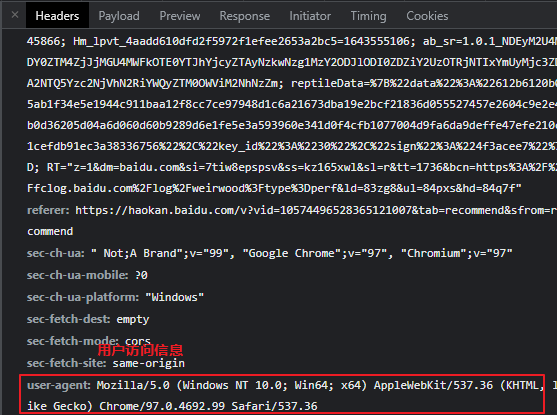
        print("爬取成功")

    except Exception:

        print("爬取失败")

最后，我们来介绍一些常见的反爬虫操作：

（1）添加headers请求头



在上例中，我们在Headers底部可找到用户访问信息user-agent ，将其封装到headers中，并打包给requests.get函数，可有效将爬虫伪装为正常的访问，代码如下：

header={

    "user-agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36"

} # 类似的，还可以添加cookie等信息，不过一般user-agent够用

Request\_URL="https://haokan.baidu.com/videoui/api/videorec?title=%E6%98%AF%E4%BB%80%E4%B9%88%E8%AF%B1%E5%8F%91%E5%BF%83%E6%A2%97%EF%BC%9F80%25%E7%9A%84%E4%BA%BA%E5%81%9A%E4%BA%86%E8%BF%99%E4%BB%B6%E4%BA%8B%EF%BC%81%E7%8E%B0%E5%9C%A8%E6%94%B9%E8%BF%98%E6%9D%A5%E5%BE%97%E5%8F%8A&vid=10574496528365121007&act=pcRec&pd=pc"

res=requests.get(Request\_URL,headers=header)

（2）降低爬虫频率

有时候，网页的服务器会因为我们爬虫行为过于频繁而封杀计算机的Ip访问地址，为了避免这种情况发生，有时候我们不得不适当降低爬虫频率，最简单的方法即是time.sleep函数，如下：

import time # 导入time 模块

for i in range(5):

    video\_url=video\_list[i]["play\_url"]

    video\_name=video\_list[i]["title"]+".mp4"

    try:

        res=requests.get(video\_url)

        new\_url=path+"/"+video\_name # 将网络链接改为本地链接

        file=open(new\_url,"wb")

        file.write(res.content) # 用二进制形式保存视频信息

        print("爬取成功")

    except Exception:

        print("爬取失败")

    time.sleep(2) # 每爬取完一次，程序暂停两秒再执行

# Excel常用操作

# Web网页制作

1基本结构

<html> <!--html源码前缀-->

    <head> <!--准备工作-->

        <title> “标题”</title> <!--网页名-->

        <style type="text/css"> <!--网页样式-->

            body {

                "css样式"

            }

        </style>

    </head>

    <body> <!--网页构建-->

        <h1> “副标题” </h1> <!--一级标题-->

        <p> <!--主体内容-->

            “主体内容1”

<br> <!—换行-->

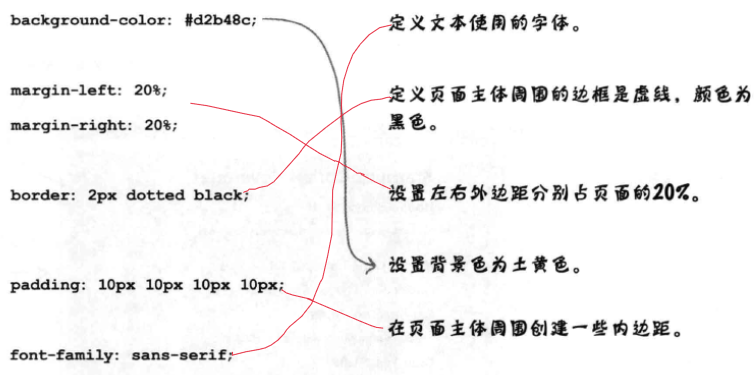
“主体内容2”

        </p>

    </body> <!--二级标题-->

</html>

2 css样式



3超链接（通过点击链接到外部网页或文件）

同级目录下链接：<a href="文件名"> “点击内容” </a>

上级目录html文件链接下级目录文件：<a href="下级目录名/文件名"> “点击内容” </a>

或 <a href="下1级目录名/下2级目录名/文件名"> “点击内容” </a>等 （下索引号）

下级目录html文件链接上级目录文件：<a href="../文件名"> “点击内容” </a>

或 <a href=".. /上1级目录文件名/文件名"> “点击内容” </a>等 （上索引号）

连接到外部网页：<a href="网页名"> “点击内容” </a>

\*提示工具：title元素

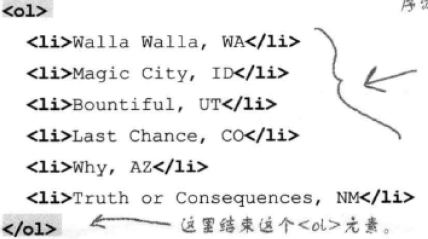
另外，可以在副标题中利用id元素创建连接目标，然后用超链接实现跳转

4引用

短引用（应用于主体内容之内） ： <q>引用内容</q> （效果为添上双引号）

长引用（应用于主体内容之外，自成一块）：<blockquote>引用内容</blockquote> （效果为另起一行并缩进）

5列表（应用于主体内容之外，自成一块）：



其中<ul> </ul>为无序列表 而<ol> </ol>为有序列表

6图像

<img scr=”图像路径/图像名”> （图像的扩展名只有三种可行，JPEG，PNG，以及GIF）

# Latex文本编辑

工具：Overleaf

1 基本架构

%以下为导言区

\documentclass{article} %文章样式

\usepackage[utf8]{inputenc} %引用宏包（中文）

\usepackage{ctex} %引用宏包（ctex编译器）

\usepackage{graphicx} %引用宏包（图形）

\usepackage{float} %引用宏包（表格）

\usepackage{cite} %引用宏包（参考文献引用）

\title{勾股定理} %标题

\author{熊高庆} %作者

\date{2021年7月21日} %写作日期

\newtheorem{thm}{定理} %定理

% 以下为正文区

\begin{document} %正文

\maketitle %标题，作者，日期生成

\begin{abstract} %摘要

\centering %居中

这是一篇关于勾股定理的小短文

\end{abstract}

\tableofcontents %目录生成

\section{勾股定理在古代} %章节一

\part{...} %第一部分

\chapter{...} %第一章

\section{...} %第一节

\subsection{...} %第一小节

西方称勾股定理为毕达哥拉斯定理，将勾股定理的发现归功于公元前6世纪的毕达哥拉斯学派\cite{Kline}。%参考文献引用

该学派得到了一个法则，可以求出可排成直角三角形三条边的三元数组。毕达哥拉斯学派没有

书面著作，该定理的严格表述和证明则见于欧几里得\footnote{欧几里得，约公元前330-275年。} %脚注

《几何原本》的命题47：“直角三角形斜边上的正方形等于两直角边上的两个正方形之和”。证明是用

面积做的。

我国《周髀算经》载商高（约公元前12世纪）答周公问：

\begin{quote} %引用

\zihao{-5}\kaishu %引用内字体变更

勾广三，股修四，径隅五。

\end{quote}

又载陈子(约公元前7--6 世纪)答荣方问：

\begin{quote}

\zihao{-5}\kaishu

若求邪主自者， 以日下为勾， 日高为股，勾股各自乘， 并而开方除之，得邪至日。

\end{quote}

都较古希腊更早，后者已经明确道出勾股定理的一般形式。图\ref{fig:xiaotu} %标签引用

是我国古代对勾股定理的一种证明\cite{Quanjing}。%参考文献引用

\begin{figure}[ht]

\centering

\includegraphics[width=5cm]{xiaotu.pdf} %引用图片

\caption{宋赵爽在《周髀算经》注中作的弦图（仿制），该图给出了勾股定理一个极具对称美的证明。} %图注

\label{fig:xiaotu}

\end{figure}

\section{勾股定理在近代} %章节二

勾股定理可以用现代语言表述如下：

\begin{thm}{勾股定理}

直角三角形斜边的平方等于两腰的平方和。

%空行用于分段

可以用符号语言表述为：设直角三角形ABC，其中$\angle ACB=90^\circ$，则有 %行内公式

\end{thm}

\begin{equation} %行间公式

\centering

AB^2=BC^2+AC^2

\label{eq:勾股定理}

\end{equation}

满足式(\ref{eq:勾股定理})的整数称为勾股数，第一节所说毕达哥拉斯学派得到的三元数组就是勾股数。

下表列出一些较小的勾股数:

\begin{table}[H] %表格环境

\begin{tabular}{|rrr|} %表格生成

\hline %表内横线

直角边$a$ & 直角边$b$ & 斜边$c$\\

\hline

3 & 4 & 5\\

\hline

5 & 12 & 13\\

\hline

\end{tabular}

\qquad

($a^2+b^2=c^2$)

\end{table}

% 参考文献引用放正文末尾

\bibliographystyle{plain} %参考文献风格

\bibliography{勾股定理.bib} %参考文献引用

\end{document}

\documentclass[a4paper]{article}

\usepackage[utf8]{inputenc}

\usepackage{ctex}

\newcommand{\song}{\CJKfamily{song}} % 正文五号字体, 单倍行距，中文默认宋体

\newcommand{\wuhao}{\fontsize{10.5pt}{15.75pt}\selectfont}

\usepackage[margin=1in]{geometry} % 设置页边距为A4样式

\usepackage{setspace}

\renewcommand{\baselinestretch}{2.0}

\usepackage{ragged2e}

\usepackage{indentfirst}

\setlength{\parindent}{2em}

\usepackage{enumerate}

\usepackage{amsmath}

\newtheorem{theorem}{Theorem}

\newtheorem{lemma}{Lemma}

\newtheorem{proof}{Proof}

\usepackage{graphicx}

\usepackage{fontspec}

\setmainfont{Times New Roman} %全文字体设置（英文）

\setsansfont{Verdana} % 常用于显眼的大标题

\setmonofont{Courier New} % 常用于排版代码程序

\usepackage{ulem}

\usepackage{lettrine}

\usepackage{float}

\usepackage{cite}

\title{笔记}

\author{熊高庆}

\begin{document}

\maketitle

\tableofcontents % 目录生成

\section{标点符号和特殊符号} %------------------------------------------------------%

中文标点符号：。 . ， 、 ； ： ？ ！ · ‘’ ”“ 《》 <> （） 【】 {} …… ——

英文标准标点符号：, . ; : ! ? ' ' () [] - / \* @

"\,'A'" % ”’A’”

A-B % A-B

A--B % A--B

A---B % A---B

$\sim$ % ~

$\dots $ % ……

$\backslash $ % \

$\prime$

\# % #

\$ % $

\% %

\& % &

\{ \} % {}

\\_ % \_

A\ B % 空格

A

B % 换行分段

A\\B % 换行不分段（不缩进）

A \quad B % 行内间隔

希腊文字符： % 大写希腊字母请首字母大写

$\alpha$

$\beta$

$\gamma$

$\delta$

$\epsilon$

$\zeta$

$\eta$

$\theta$ \quad $\Theta$

$\lambda$

$\mu$

$\xi$

$\pi$

$\rho$

$\sigma$

$\tau$

$\upsilon$

$\phi$

$\varphi$

$\chi$

$\psi$

$\omega$

罗马数字

$\uppercase\expandafter{\romannumeral1}$ % 大写

$\romannumeral1$ % 小写

\section{字体和强调} %------------------------------------------------------------%

英文正文默认罗马字体，直立，中等。

字体：

\textrm{A} % 罗马字体 或写{\textrm xxx}

\texttt{A} % 打印字体

字形：

\textup{A} % 直立体

\textit{A} % 意大利体

\textsl{A} % 倾斜体

字宽：

\textmd{A} % 中等体

\textbf{A} % 加粗体

组合：

\texttt{\textit{\textbf{A}}} % 组合

中文如下（中文默认宋体）

\songti{国} % 宋体

\heiti{国} % 黑体

\fangsong{国} % 仿宋

\kaishu{国} % 楷书

现代用法：引用fontspec（英文）或xeCJK（中文）宏包，在导言区预设定字体，如下：

$\backslash$setmainfont\{Tmines New Roman\}

$\backslash$setCJKmainfont\{kaishu\}

文字强调： % 需引用ulem宏包

\uline{ABCD}

\uuline{ABCD}

\uwave{ABCD}

\dotuline{ABCD}

\section{排版} %-----------------------------------------------------------------%

纸张大小：A4（见导言区设置）

（1）字号：

{\tiny A} % 小六号（注意不采用\tiny{A}这种写法，会改变后面所有字体）

{\scriptsize A} % 六号

{\small A} % 五号

{\normalsize A} % 小四号

{\large A} % 小三号

{\Large A} % 小二号

{\LARGE A} % 二号

{\huge A} % 小一号

（2）对齐：

\begin{flushleft} % 左对齐

A

\end{flushleft}

\begin{center} % 居中

A

\end{center}

\begin{flushright} % 右对齐

A

\end{flushright}

环境里面常使用居中命令，如下：\\

$\backslash$begin\{xxx\}\\

$\backslash$centering\\

xxxxx\\

$\backslash$end\{xxx\}

\justifying{123} % 需引用宏包ragged2e

（3）下沉效果： % 引用宏包lettrine

\lettrine{西}方称勾股定理为毕达哥拉斯定理，将勾股定理的发现归功于公元前6世纪的毕达哥拉斯学派

（4）行间距

全局变化： %\renewcommand{\baselinestretch}{1.1} 在导言区设置

局部修改：

123\\

123

123 \vspace{2cm} \\

123

（5）手动设置首行缩进 %需引用宏包indentfirst

\indent{123} % 缩进

\noindent{123} % 不缩进

\hspace{2em}{123} % 不得已的缩进办法

（6）页码设置

本页无页码，后续页码从下一页开始！！！

\thispagestyle{empty} % 当前页无页码

\newpage

\setcounter{page}{5}

\section{公式} %-----------------------------------------------------------------%

（1）行内公式：

$a+b=c$ % 行内公式（内嵌公式）

（2）行间公式：

\begin{equation}

a+b=c \quad \text{当}a,b>=0 % 公式中插入文字

\end{equation}

或者:

\begin{subequations} %需引用宏包amsmath

\begin{equation}

a+b=c

\end{equation}

\begin{equation}

a-b=d

\end{equation}

\end{subequations}

（3）常用公式符号：

常用数学符号：

$\le$ % 小于等于

$\ge$ % 大于等于

$\neq$ % 不等于

$\in$ % 属于

$\notin$ % 不属于

$\rightarrow$ % 右单箭头

$\Rightarrow$ % 右双箭头

$\leftrightarrow$ % 双向单箭头

$\Leftrightarrow$ % 双向双箭头

$\mathrm{d} y$ % 导数

$\partial y$ % 偏导数

$\ln x$\quad$\log x$\quad$\log\_{a}{b}$ % 对数

$\int\_{a}^{b} x^2 \mathrm{d} x$ % 积分

$\sum\limits\_{i=1}^{n}{a\_{i}}$ % 求和

$\lim\limits\_{i\to +\infty}{a\_{i}}$ % 极限

$\prod\limits\_{i\to n}{a\_{i}}$ % 连乘

上标与下标：

$A\_{ij}=2^{i+j}$ %上标与下标，\_表示下标，^表示上标，之后引用的内容用{}括起来

$\sum\limits\_{i=1}^{n}{i^2+3i+4}$

$\lim\limits\_{i\to +\infty}{i^2+3i+4}$

$\prod\limits\_{ii\to +\infty}{i^2+3i+4}$

$\overline{X}$

$\hat{X}$

$\widetilde{X}$

$\dot{X}$

分式：

$\frac{a+b}{\frac{a}{b}+c}$

根式：

$\sqrt{x^2+2x+1}$

矩阵：

$A=\begin{matrix}

a\_{11} & a\_{12} & a\_{13}\\

a\_{21} & a\_{22} & a\_{23}\\

a\_{31} & a\_{32} & a\_{33}

\end{matrix}$

定界符：

$\lim\limits\_{x \to 0}\left(\frac{a^x+b^x+c^x}{3}\right)^{\frac1x}$ %left和right用于定界

多行公式排版1：

\begin{align}

&\lim\limits\_{x\to 1}\left(\frac{1}{1-x}-\frac{3}{1-x^3}\right) \notag\\

= &\lim\limits\_{x\to 1}\left(\frac{x^2+x-2}{1-x^3}\right) \notag\\

=&\lim\limits\_{x\to 1}\frac{(x+2)(x-1)}{(1-x)(x^2+x+1)} \notag\\

=&\lim\limits\_{x\to 1}\frac{-(x+2)}{x^2+x+1} \notag\\

=& -1

\end{align}

多行公式排版2：

\begin{equation}

\begin{cases}

a+b=c\\

a-b=d\\

\end{cases}

\end{equation}

\section{其他元素} %---------------------------------------------------------%

（1）列表：

\begin{itemize}

\item This is the first row.

\item This is the first row.

\item This is the first row.

\end{itemize}

或者：

\begin{enumerate}[A.] %需加载宏包enumerate，自定义过程需要将内容加在后面的[]中。

\item hello

\item hello

\item hello

\end{enumerate}

（2）表格：

\newpage % 避免表格图片等插在文字中间，干脆直接跳页

\begin{table}[h!]

\centering

\begin{tabular}{|c|c|c|} % 竖线“|”表示表格对应列内加竖线

\hline %头行线

x & y & z \\

1 & 123 & 23 \\

\cline{2-3} % 表内行线，也可用\hline

34 & 57 & 789 \\

\hline %尾行线

\end{tabular}

\caption{my first table}

\end{table}

（3）图片：

\begin{figure}[htb] %需引用宏包graphicx，若引用宏包float，则要是写[H]，图片位置会固定

\centering

\includegraphics[scale=0.3]{chapter2-1.jpeg}

\caption{my first figure}

\end{figure}

%width=\* 宽度

%height=\* 高度 高度和宽度必须标明明确的单位，比如厘米（cm）或者英寸（in）

%scale=\* 倍数

%angle=\* 顺时针旋转角度

（4）摘要：

% 英文摘要

\begin{abstract}

This is a demo\\

\textbf{keywords：} Latex

\end{abstract}

还可以通过在导言区加用下列命令来修改摘要：

% \renewcommand{\abstractname}{\large Abstract}

% 中文摘要

\flushleft{\textbf{摘要：} 这是一个demo}

\flushleft{\textbf{关键词：} Latex}

（5）定理与证明：% 需引用宏包amsmath

\begin{theorem}

This is a theorem.

\end{theorem}

\begin{lemma}

This is a lemma.

\end{lemma}

\begin{proof}

This is proof.

\end{proof}

（6）参考文献引用

step1：编辑xxx.bib文件，如myref.bib

step2：在需要引用参考文献的文字后加上这段代码\cite{valentin2016learning} % 所加内容即为article标题（不可为中文）

step3：引用宏包cite，并在文件末尾加上：

\bibliographystyle{unsrt} % 按引用次序编排参考文献

\bibliography{myref}

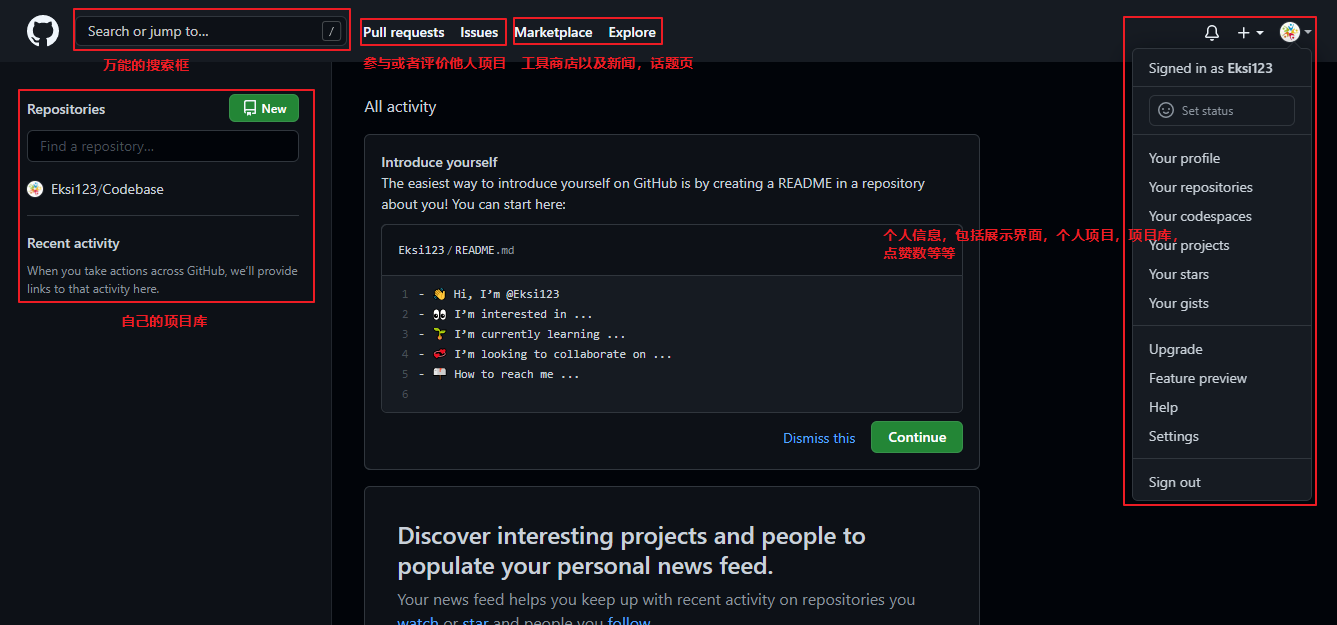
\end{document}

# Github项目库

\* 鉴于Github是国外网站，我们需要借助第三方工具“dev-sidecar”来托管访问，访问时切勿关闭该软件。此外该软件还可托管访问诸如Stack Overflow，dns等

\* 现阶段我们暂时不需要使用到Git的历史代码记录功能，而仅仅是用Github来托管保存自己的项目代码，以及搜寻他人优秀项目这一功能，故不介绍Git

1 Github界面介绍



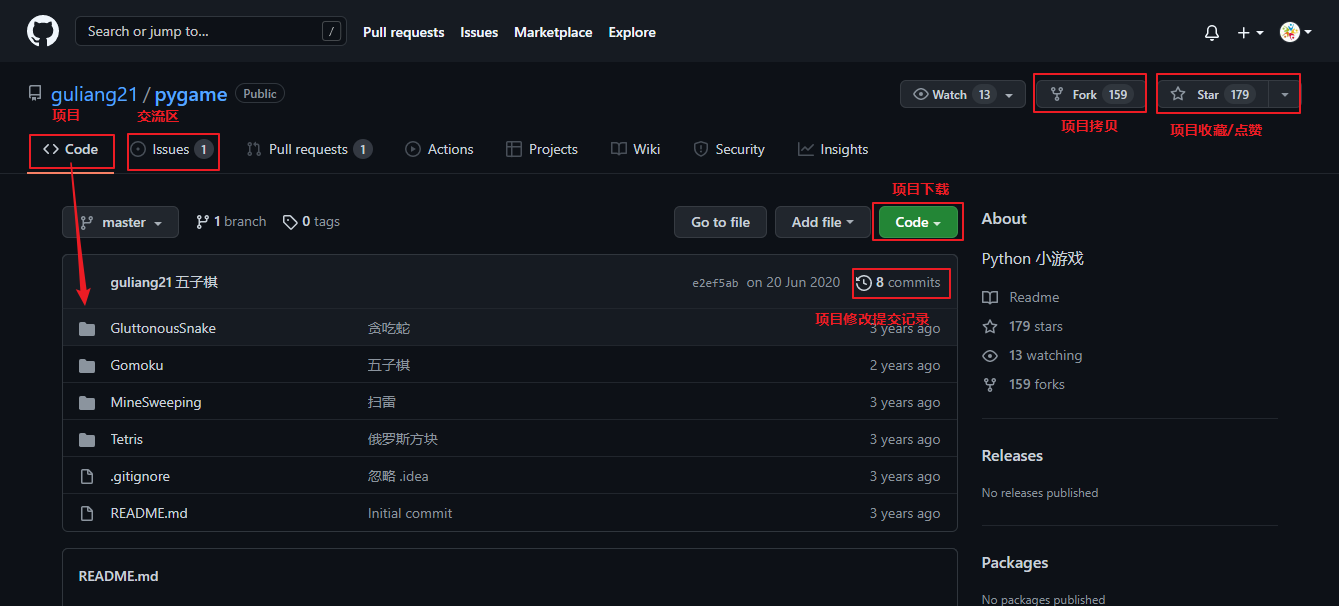
其中我们用得较多的也就是搜索框和个人项目库，分别对应了我们的两个目的：寻找，收藏并下载他人项目，以及管理自己的项目，下面我们来逐一介绍：

2 寻找，收藏并下载他人项目



如上所示，输入Python小游戏，我们可左半框的结果，其中包括repositories项目库，code代码，以及commits提交结果，issues提问，discussion讨论等等，

以项目库为例，我们点击第一个

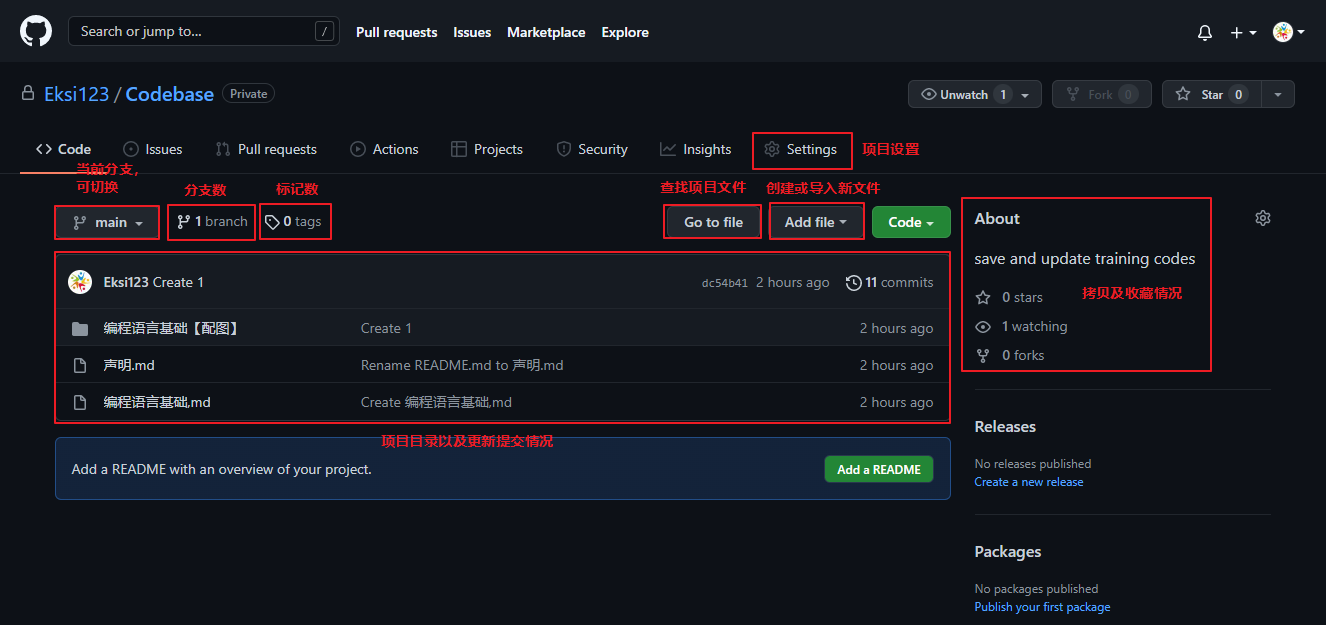


其中Code为项目区，可点击任一项目进行查看，Issue为问题交流区，fork用于拷贝该项目库为己用，star用于收藏该项目，commits用于查看项目修改记录

Code用于下载该项目至本地，如下：

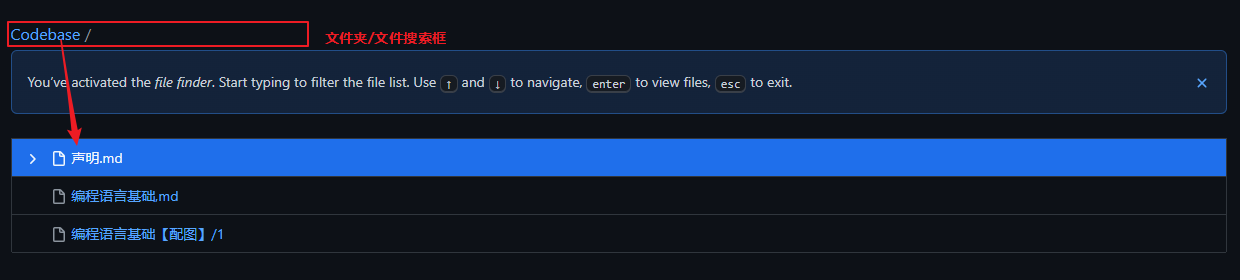


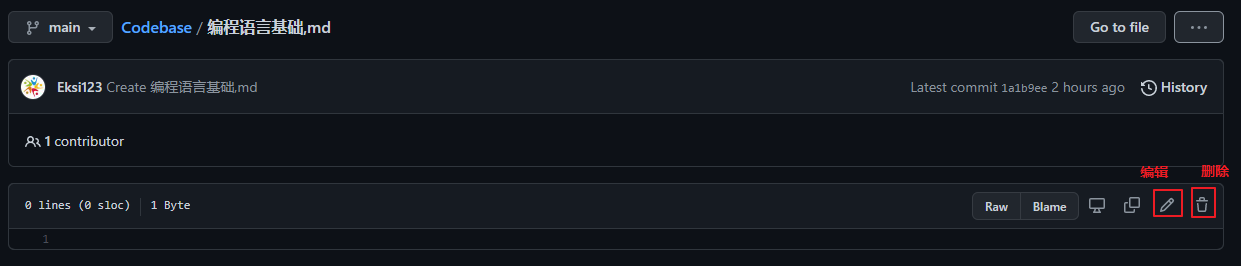
3 管理个人项目

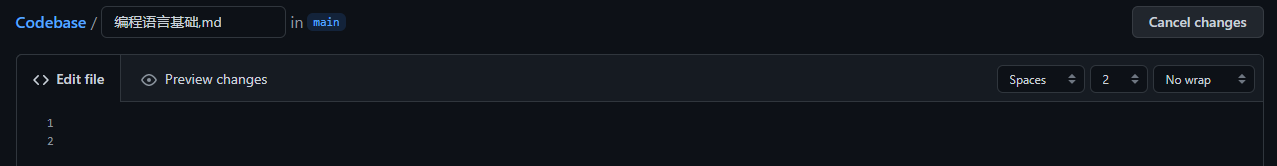


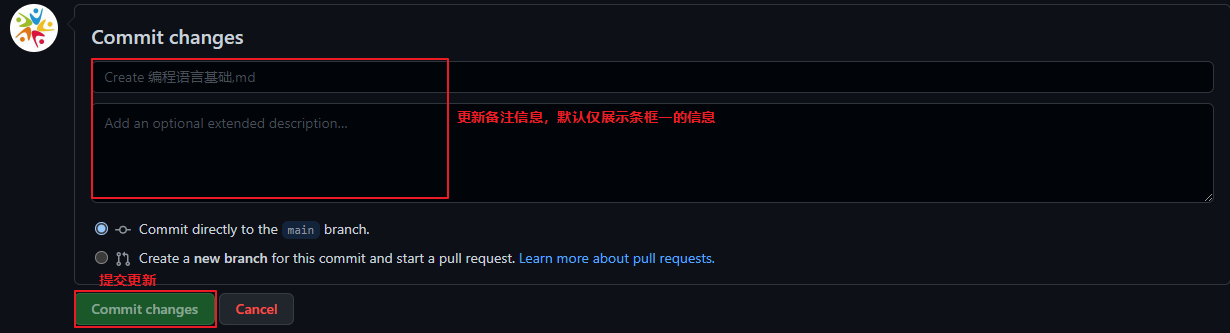
Github项目库中的项目结构为branch -> directory -> file对于我们来说，主分支main就足够使用了，我们可以在该分支下创建，导入文件夹以及文件。在项目目录区域我们可查看各文件夹/文件最近的更新情况，点击右上角的11 commitis我们可查看所有文件夹/文件的历史更新情况。右侧的stars和forks即为别人的收藏，拷贝数。下面我们着重来介绍创建，导入，修改文件夹/文件以及更新备注功能

3.1 修改文件夹/文件及更新备注信息



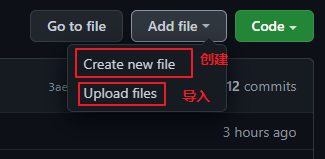


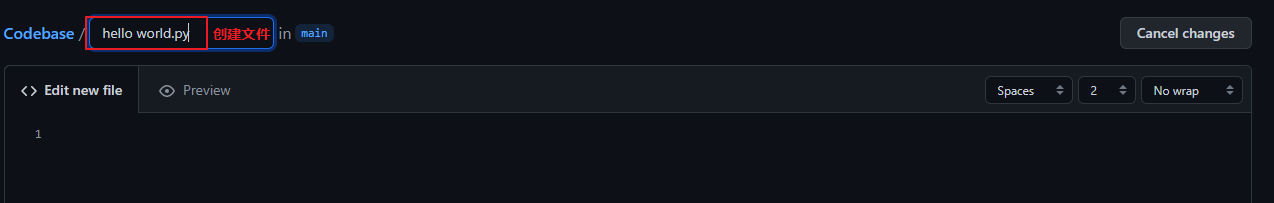


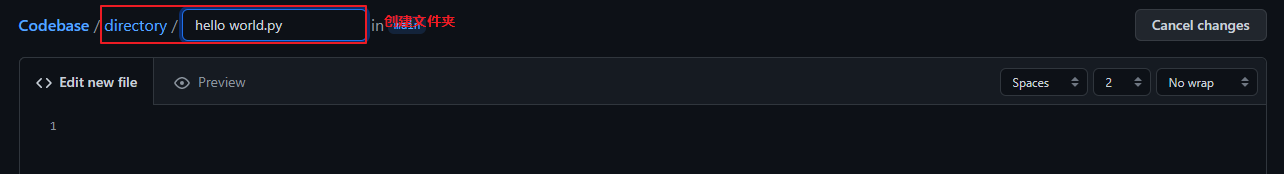


点击 go to file，我们以编程语言基础.md文件为例，在点击编辑之后，即可在Github自带的编辑器中进行内容编辑，最后在底部添加更新备注信息并提交更新，更新备注信息可以是关于内容修改的细节信息，也可以是方法，思路，功能的变更等等，目的在于便于后期项目回溯。

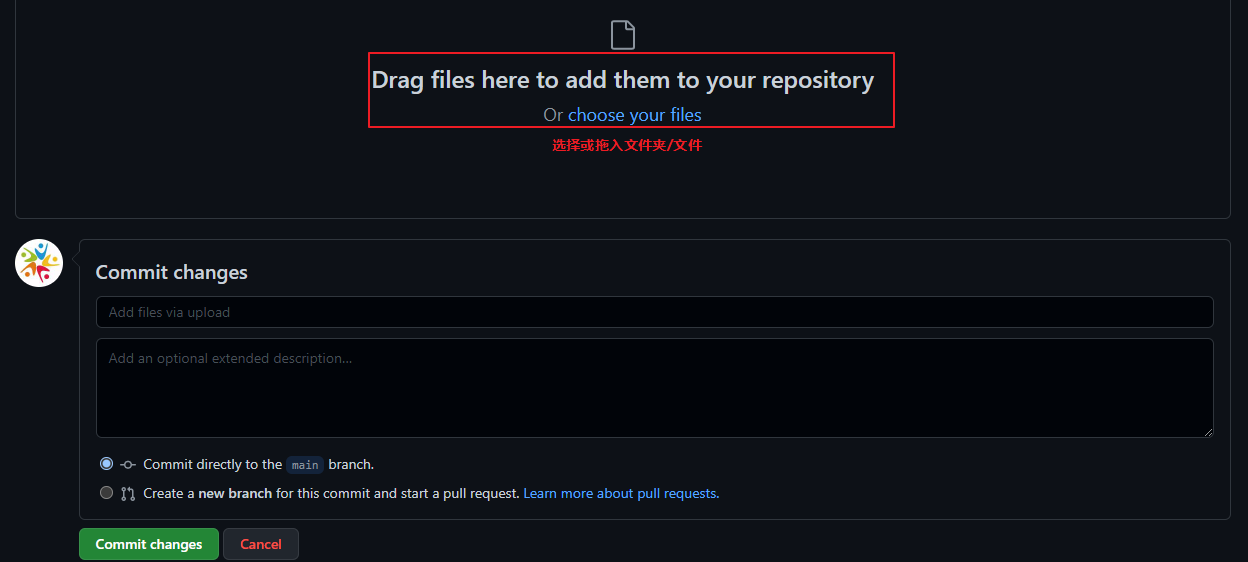
3.2 创建及导入文件夹/文件







点击create new file便可创建文件夹/文件，其中输入“文件名”为创建文件，输入“文件夹名/”为创建文件夹（注意不可创建空文件夹）。创建之后在下方便可以编辑（当然除markdown文件外一般我们不在这里编辑），最后在底部可更新提交。



点击 upload file便可如上直接导入文件。

4 markdown文本标记

不同于latex，markdown仅仅作为一种简单的文本标记语言来使用（类似html），在Github中为了便于在自己的项目库中展示笔记，说明等，我们需要使用markdown。本文只简单介绍Markdown的最基本功能：标题大纲，加粗斜体，分割线，符号标记，插入图片，插入表格，插入超链接，插入代码块。我们举例如下：

markdown常用命令如下：

\*\*\* \*\*\*用作分割线

【标题大纲】

# 一级标题 #+标题名，下同

## 二级标题

### 三级标题

\*\*\*

【加粗斜体】

\*倾斜\* \*文字\*，下同

\*\*加粗\*\*

\*\*\*倾斜加粗\*\*\*

\*\*\*

【符号标记】

符号前面加转义符"\"即可，如下：

\\*

\-

\\_

\#

\()

\*\*\*

【插入图片】

插入项目库内图片

![](/timg5.png) ! [图片描述，可不加](相对路径+ /+ 图片名)

插入网络图片

![](<http://img0.iplant.cn/image61/b/1633320.jpg>) ![](图片网络路径)

用html语言插入图片，可调整大小

<img src="https://github.com/Eksi123/Codebase/blob/main/timg5.png" width="50%">

\*\*\*

【插入表格】

插入列表

- 第一行

- 第二行

- 第三行

插入表格

学号|姓名|成绩

-|-|-

01|小明|90

02|小红|88

03|小王|85

\*\*\*

【插入超链接】

链接到网站

<https://github.com/Eksi123/Codebase> <网址>

链接到项目库内文件

[file](/README.md) [文件描述，一定要加](相对路径+ / +文件名)

\*\*\*

【插入代码块】

``` 英文半角符号```xxx```，xxx为代码块

for i in range(5)

print(i)

```

# Mysql数据库

Mysql+Redis

1登录数据库

Cmd控制台：mysql -uroot -p (然后enter回车) 再输入密码

可以在cmd控制台运行或vscode 依次输入 host , user, password, port运行

2展示数据库，表和表的列

（1）展示数据库：show databases;

（2）展示表：use 数据库名; show tables;

（3）展示表的某一列：show 列名 from 表名;

展示表的所有列：DESC 表名

3表的建立，删除与导入导出

（1）创建表

USE 数据库名;

CREATE TABLE customers IF NOT EXISTS(

Id INT NOT NULL AUTO\_INCREMENT,

name char(100) NOT NULL DEFAULT xxx,

address char(100) NULL,

PRIMARY KEY (id)

);

（2）删除表

DROP TABLE 表名

（3）插入列

INSERT INTO customers(name, address)

VALUES('ny','JiangXi');

（3）从外部导入对应表

load data infile”'路径及文件名 如：D:/matcars.csv”

into table 表名 (如果包含中文，则加上 character set gbk

　　fields terminated by ',' optionally enclosed by '"' escaped by '"'

lines terminated by '\r\n';

4调用mysql数据库

\*Redis数据库简单介绍：

（1）Redis数据库与Mysql数据库区别：

首先，两者均为数据库，Mysql为关系型数据库（表格型，行列定位）而Redis为非关系型数据库（键值型，行定义）；其次，Mysql一般用于海量数据存取，故数据放在磁盘中，而Redis用于少量且常用的数据存在，故数据放在“缓存”里（读取到内存很快）

（2）登录数据库

Cmd控制台

cd C:\Redis-x64-3.0.504

redis-server.exe redis.windows.conf

打开新的cmd控制台

cd C:\Redis-x64-3.0.504

redis-cli

或者直接在vscode上连接，可查看数据集

（3）键值输入输出

set key “value”

get key