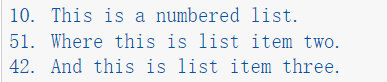
Python A -- Course Content Summary

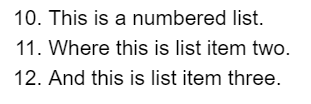
# Module A

## Markdown

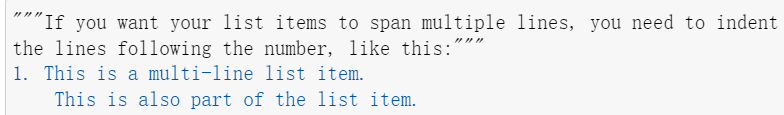
* \*\*\* \*\*\* italic and boldface
* # ## ### {-}?
* ordered list

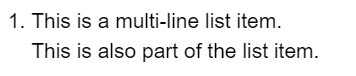


it will show below:

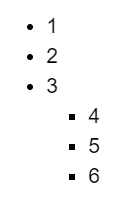


* span multiple lines





* unordered list



* Block Quotes >
* Verbatim text ???
* Link

This is a link to [my blog](http://www.mailund.dk).

others???

* image

![Title of the figure](URL-to-figure)

![Title of the figure](full path to/this\_figure)

![local picture](C:\Users\T480S\Desktop\R\_study\R\_shortcuts.png) ？？？

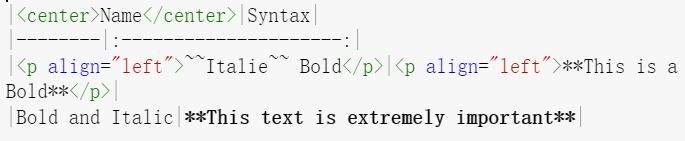
* horizontal rules \*\*\*

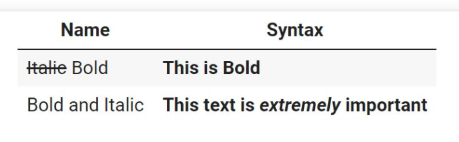


* code block ””” ””” note: distinguish this triple quotation from that one under the edit mode



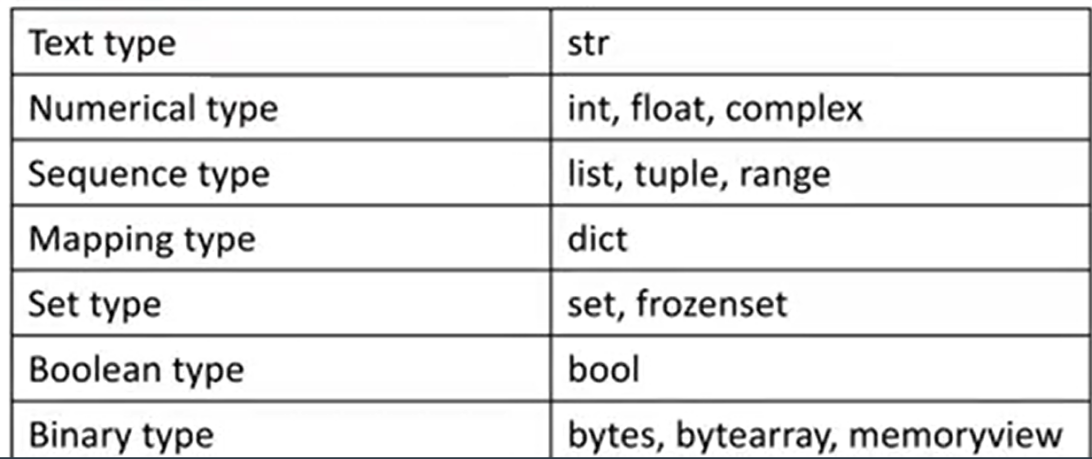
* Build table

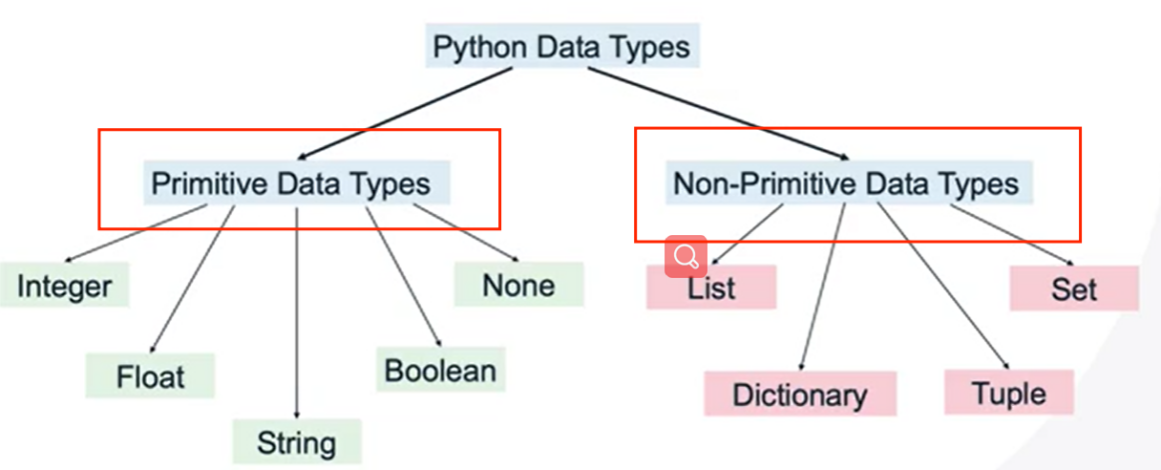




## 1.2 Variable

* Data types





tip:

(1)from negative infinity to infinity

(2)only one type of integer

(3)a, b, c = 10, -5, 0 or a=10;b=-5;c=0

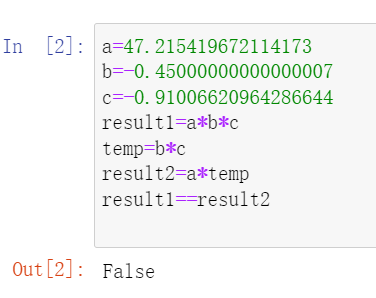
(4)unary operation +=

(5)precedence associativity

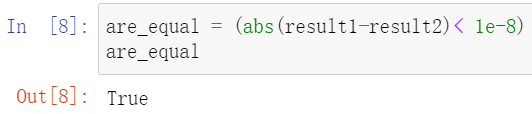
(6) \*\* is the highest priority. // % / \* is the second priority. +- is the lowest one

* roundoff error

round(3.1415,2) output 3.14



solution: set a threshold to detemine equality



## 1.3 math & cmath (complex literial)

* from math import \*
* dir() dir(math) shows all the functions in the math library
* factorial(3) 3!
* logyx math.log(x,y)

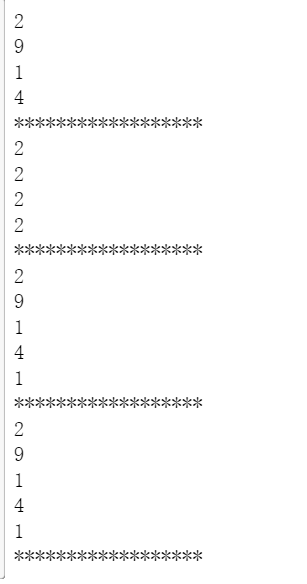
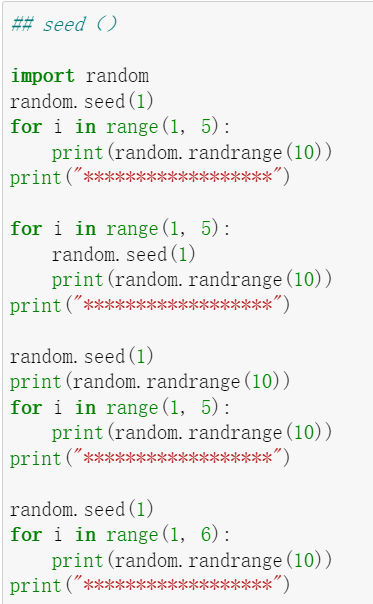
math.log(math.e) = 1 natural logarithm of the input argument math

也就是log()=ln()

* math.exp(x) math.log10(x) math.sqrt(x)
* math.ceil(x) math.floor(x)
* cmath.sqrt(-1)
* 1+2j == complex(1,2)
* decimal\_part,int\_part= math.modf(q8\_num)

separate a number into fractional and integer part

## 1.4 random



* random.uniform(-3.5,8.1)
* random.randint(-7,10) containing 7 and 10
* random.randrange(5,7,2)

random.randrange(start, stop, [step]) stop not inclusive

* random.choice([list]) select random elements from list ,tuple also be ok
* random.choices(a,k=6)
* random.choices(a,weights=[1,1,1,1,1],k=6), the weights of every element
* random.random() [0,1) float type

## 1.5 string

* note the difference between print(multipleStringLine) and multipleStringLine “““ ”””
* len(*string*) include blank spaces
* str1 = “bye! ”\*3
* build-in function lower(),upper(),replace(x,y),count(x),isalpha(),capitalize(),title()

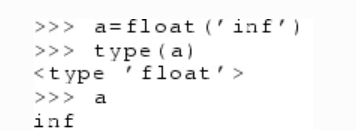
in, not in, swapcase()

* string.strip() .lstrip() .rstrip() #default leading character is blank space

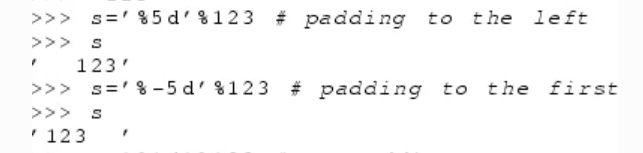
txt = ",,,,,rrttgg.....banana....rrr"  
 x = txt.strip(",.grt")  
 print(x)

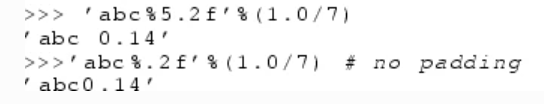
output: banana

* look for all the build-in function for string, stringvaribale. tab
* str1[0], str1[-1],str2[-3],str[1:2],s[-2:-1] = s[-2]
* str[2:2] or str[2:1] will give the result of ‘ ’
* str1[1:5:2] or str1[5:2:-2] the third arg is step
* Two special strings that can be converted to a floating point numbers are ‘inf’ and ‘-inf’, which represent infinity and negative infinity, respectively



* the first% indicates the pattern start, 5 and -5 is left or right of string, the second% symbol is the argument to fill the pattern





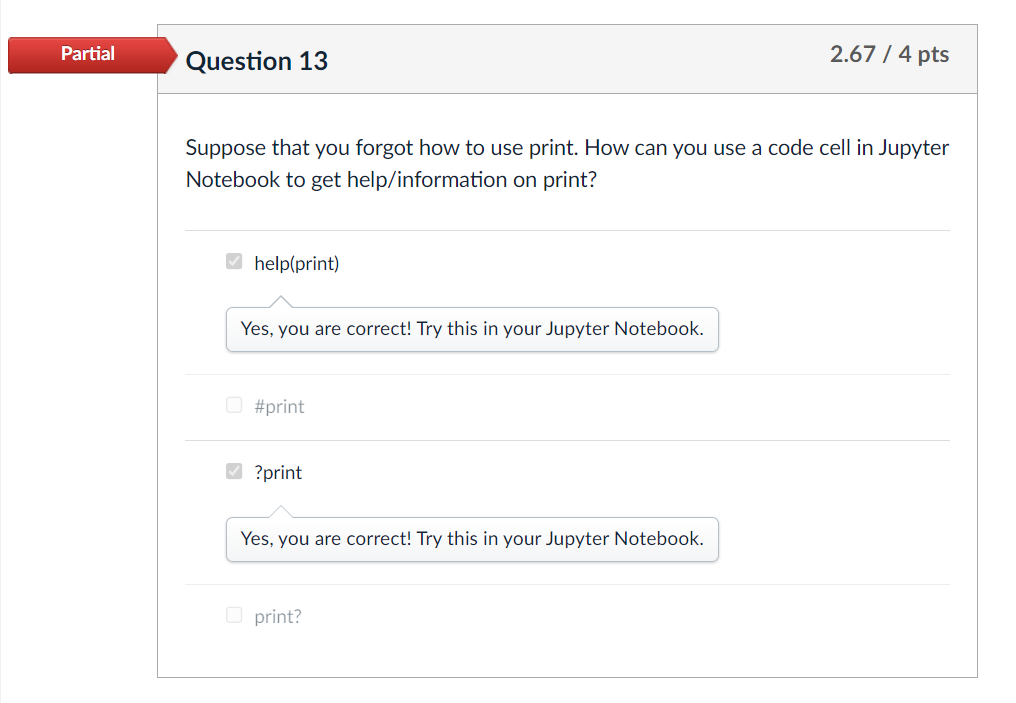
## 1.6 Quiz

(4) alt+ shift+ ctrl+ Enter

(11) Multi-line comments start with /\* and end with \*/ it’s not right

help() ?math math?

(13)



Question 20

What is the output of the following print() statement?

#named placeholders

txt1 = "My name is {fname}, I am {age}".format(fname = "John", age = 9)

#empty placeholders

txt2 = "My name is {}, I am {}".format("David", 12)

#price inside named placeholder

txt3 = "The price of John's toy is only ${price:.2f}".format(price = 49)

#price inside empty placeholder

txt4 = "The price of David's toy is only ${:.2f}".format(99.1212)

print(txt1, txt2, txt3, txt4, sep = '. ')

Note that the format() formats the specified values and insert them inside the given placeholders. The placeholder is defined using curly brackets: {}.

Q23



## 1.7 assignment

## 2.1 = cascaded

person1\_age = 50

person2\_age = 30

same\_age = person1\_age=person2\_age

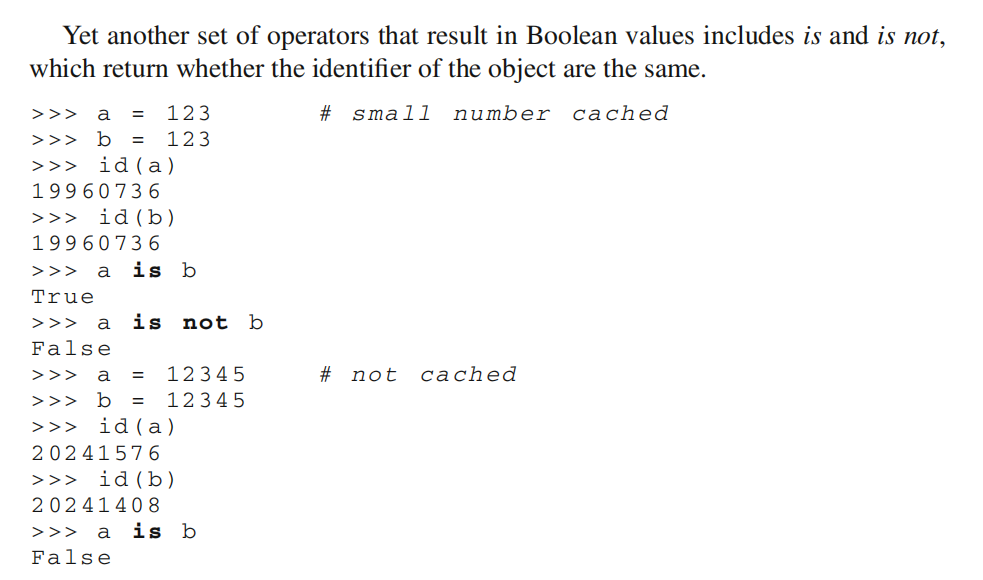
print(same\_age) output 30

## 2.2 logic expression

* a>b and b>c or b>c and c<a

and has higher priority than or

* 1 < 2 < 3 is interpreted as 1 < 2 and 2 < 3







## 2.3 eval(“1+1”) output :2

eval() parse the expression within the parenthesis and run it in the python.

## 2.4 isinstance()

isinstance(3, (int, float))

returns True if the object is an instance of the class or of a subclass of the class. It can also take a tuple of classes, in which case it returns True if the object is an instance of any of the classes in the tuple.

## 2.5 assert

* assert user\_input.isdigit() == True, "User's input is not numbers"

Python isdigit() 方法检测字符串是否只由数字组成，只对 0 和 正数有效。

* assert isinstance(user\_input, int), "Have not converted the input() to int"

## 2.6 logging

logging.basicConfig(level=logging.INFO, format='%(asctime)s-%(filename)s[line:%(lineno)d]-%(levelname)s:%(message)s',

#datafmt = '%H:%M:%S',

filename='LOG//workshop.log',

filemode='w')

* the style of LOG message

{LEVELNAME}:{NAME}:{MESSAGE}:{ASCTIME}:{FILENAME}:{FUNCNAME}:{LINENO}

* logging.DEBUG output : 10 get relevant log level number

logging.levelToNumber[10] output: DEBUG convert relevant level number to level name

## 2.7 Assignment

question8: decimal\_part,int\_part= math.modf(q8\_num)

question15: equilateral, isosceles or scalene.

question17: isinstance(x,int)

isinstance(q17\_variable,complex)

isinstance(my\_value, type(None))

# 3 While Loop

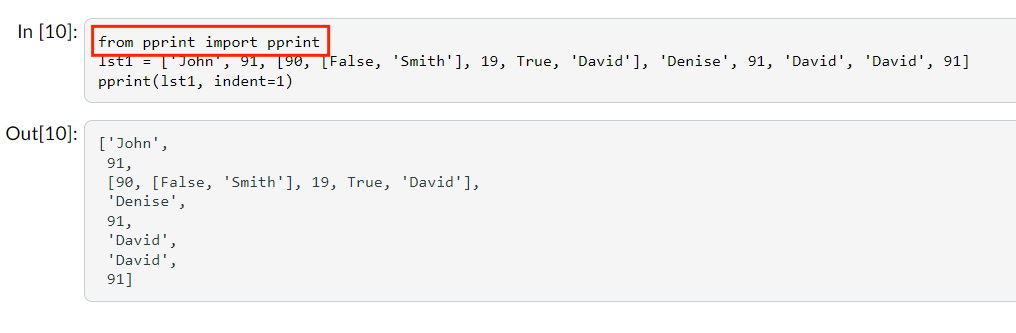
strip()删除首尾字符

lstrip(‘a’)删除左侧带有的a的字符

rstrip(‘\n’)删除右侧的\n

# 4 list and dictionary

## 4.1 list



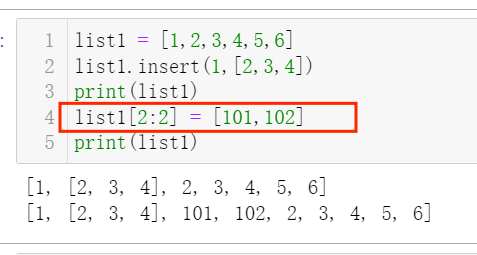
增操作：

list1+list2

list1\*2

list.insert(index,[1,2,3])一次只能增加一个操作在相应的index，如果想要插入多个元素

my\_list = my\_list[:2] + [6, 7, 8] + my\_list[2:] 或者如下所示。



append() and extend(), append(“element”) argument is element, list1.extend(list2) argument is list, list3 = list1+list2 it will creates a new list, extend()will change original list.

删操作：

1. list.remove() remove the first matching element in a list, If the element is not in the list, the method returns an error.

#while方法去除“”值

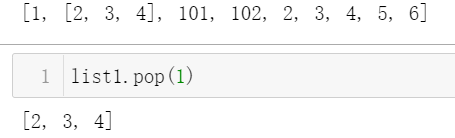
q13\_list =[37, 60, 41, 30, "", 15, 59, "", "", 76, 79, 91]

while "" in q13\_list:

q13\_list.remove("")

1. list.pop() it will remove the last element in the list and return the element.

list.pop(1) remove and return an item at a specified index



pop() method can be used with a dictionary to remove and return a value associated with a specified key dict.pop(“key”)

1. del list[1], del list[1:3], del list
2. list.clear()

查操作

1. 一个元素是否在list

element in list1; not in

1. list[1] 与list.index(1)区别

前者是查询1位置的元素是多少，后者是查询元素为1的数 在list中的位置是多少。

1. list.copy()
2. sort()

.sort()方法会直接对原始列表进行排序，并将其改变为已排序的列表。如果需要原始列表保持不变，可以使用sorted()函数来创建一个新的已排序的列表。

需要注意的是，sorted()函数适用于所有可迭代的对象，而.sort()方法只适用于列表。

d = {'a': 1, 'c': 3, 'b': 2}

sorted\_dict = dict(sorted(d.items(), key=lambda x: x[0]))

print(sorted\_dict)

lst = [('Earth', 149.6), ('Jupiter', 778.5), ('Mars', 228.0), ('Mercury', 57.9), ('Neptune', 4515), ('Saturn', 1432), ('Uranus', 2867.0), ('Venus', 108.2)]

# 使用 lambda 函数指定排序关键字为第二个元素,由大到小排列。

lst.sort(key=lambda x: x[1],reverse = True)

key的前提是针对list里的元素说的，这一题lst的element是tuple，那么排序的主体就是tuple，也就是lambda x 里的x所表示的对象，x[1]就是根据tuple第二值进行排序。

同理lst[‘key’,’wwe’,’Kkk’]

lst.sort(key=str.upper,reverse = True)

lst元素是string，那么排序主体就是string，所以key=str.upper.

a = lambda x：x+1 a(10) = 11

1. list.reverse() list = list[::-1]
2. list.count(“string”) 计算个数
3. lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# 使用lambda表达式定义过滤函数

filtered\_lst = list(filter(lambda x: x % 2 == 0, lst))

print(filtered\_lst)

# [2, 4, 6, 8, 10]

需要传入一个函数和一个序列作为参数，函数用于对序列中的每个元素进行判断，如果返回值为True，则保留该元素，否则过滤掉该元素。

## 4.2 dictionary

Initialization of two style

age1 = {"John":52, "Siobhan":21, "Ye":18}

age2 = dict(John=52, Siobhan=21, Ye=18)

key type(s) you choose must be immutable.

增操作：

my\_dict = {'a': 1, 'b': 2}

# 添加新的键值对

my\_dict['c'] = 3

my\_dict.update({’c’:3})

# 使用字典推导式添加多个键值对

new\_dict = {k: k.upper() for k in ['c', 'd', 'e']}

# 将新的键值对合并到原始字典中

my\_dict.update(new\_dict)

dict1 = {'a': 1, 'b': 2, 'c': 3}

dict2 = {'d': 4, 'e': 5, 'f': 6}

zipped = zip(dict1.keys(), dict2.values())

print(dict(zipped)) 这里zip只是两两配成元祖，前面可以加dict，list，tupe，list（zipped）都可以转换为相应类型。

删操作：

del dict[‘a’] pop[‘a’] 都能删除键值对，但是pop返回值

#删除多个键值对

keys\_to\_delete = ['a', 'c'] # 生成新字典，不包含要删除的键值对

new\_dict = {key: value for key, value in my\_dict.items() if key not in keys\_to\_delete}

dict.clear()

查操作：

查一个key是否在dict里 if key in dict not in

查一个value是否在dict里 if ‘Alice' in dict.values() not in

.keys() .values() 返回的是列表

.items()返回的是键值对元祖的列表

改操作：

repr( ) returns the string representation of the dictionary

repr(dictionary)

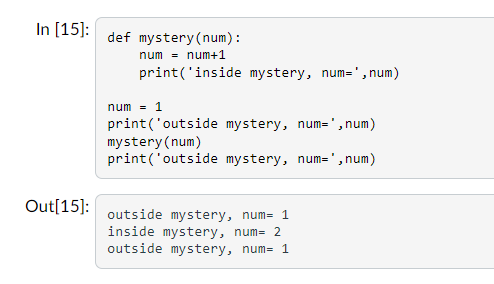
# 5 function

## 5.1 scope of variable

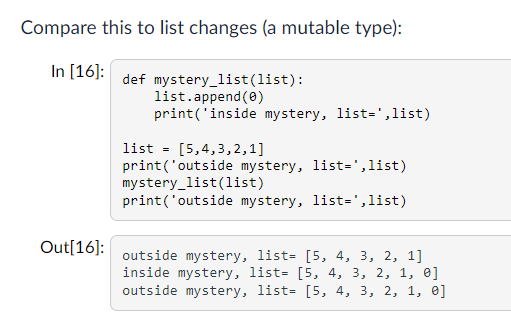
* What happens if you have the same variable name in the main body of the program and also inside of a function? See the example below. In such instances, Python will treat them as two separate variables; one in the main body of the program belongs to the global scope and the other inside the function belongs to the local scope.
* Python has the global keyword, if you want to make a variable global. Therefore, you can use the global keyword in a function body to create a global variable to make it accessible outside of the function

## 5.2 side effects in function

* if the input parameter is immutable type, the function will create a new local variable inside function, but the same name variable outside the function will never be changed.



* however, if the input parameter is mutable type ,like list, the list variable will be changed outside.



def swap\_list(list1, list2):

temp = list1

list1=list2

list2=temp

list1=[1,2,3]

list2=[4,5,6]

swap\_list(list1,list2)

print(list1,list2)

output: [1, 2, 3] [4, 5, 6]

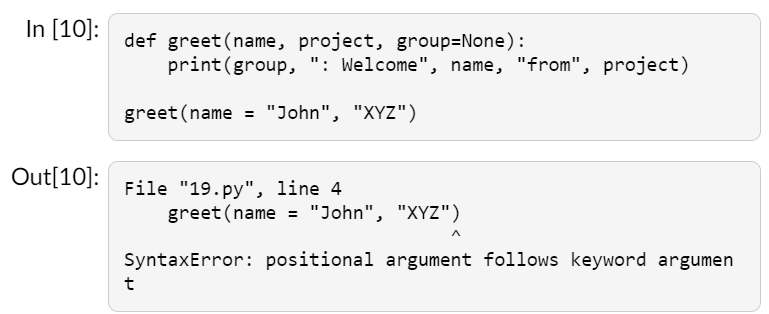
尽管上个例子list.append(0),改变了outside list的value。但是下边代码尽管参数是可变的list类型，用“=”赋值操作依然不会改变原数值，因为使用“=”就会函数内部创建local value。Notice that in the mystery\_list function there is no assignment. There is no '='. The use of '=' creates a new variable local to the function.结论： function should only access values from the parameters or change only mutable parameters using the functions (rather than assignment)

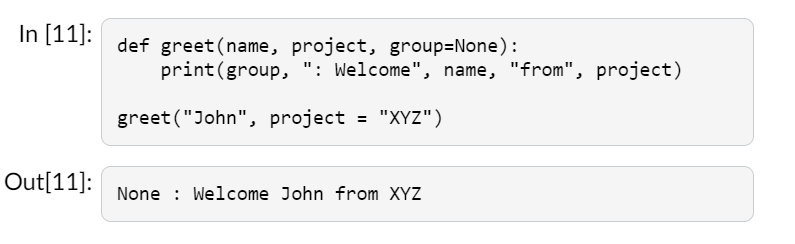
## 5.3 parameter and argument

* 1. parameter是指函数定义中参数，而argument指的是函数调用时的实际参数。

1. 简略描述为：parameter=形参(formal parameter)， argument=实参(actual parameter)
2. Parameters (values passed to a function) Arguments (the values used when actually calling the function, that is assigned to the function’s parameters)
3. you will have to arrange all your default parameters after non-default parameters.
4. Keyword Arguments:

keyword arguments, they do have to come after positional arguments and before default arguments in a function call





## 5.4 Variable Number of Arguments

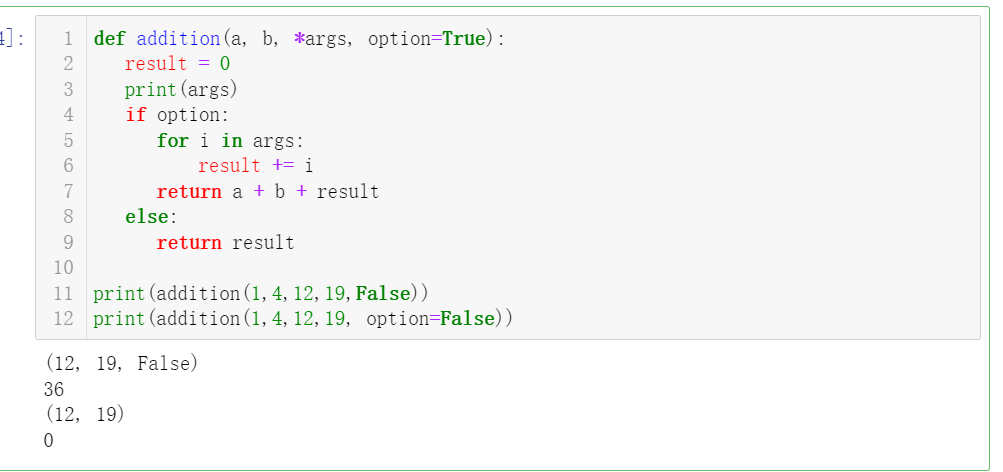
* The single \* will place all of the values into a *tuple* (It is an immutable list)

The second option (\*\*) will place all of the values into a *dictionary*

*return args -> tuple*

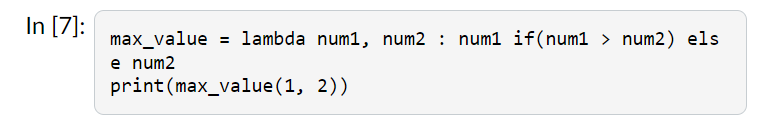
*return kargs -> dictionary*

位置参数在前，关键字参数在后，然后是默认参数，最后是可变长参数。

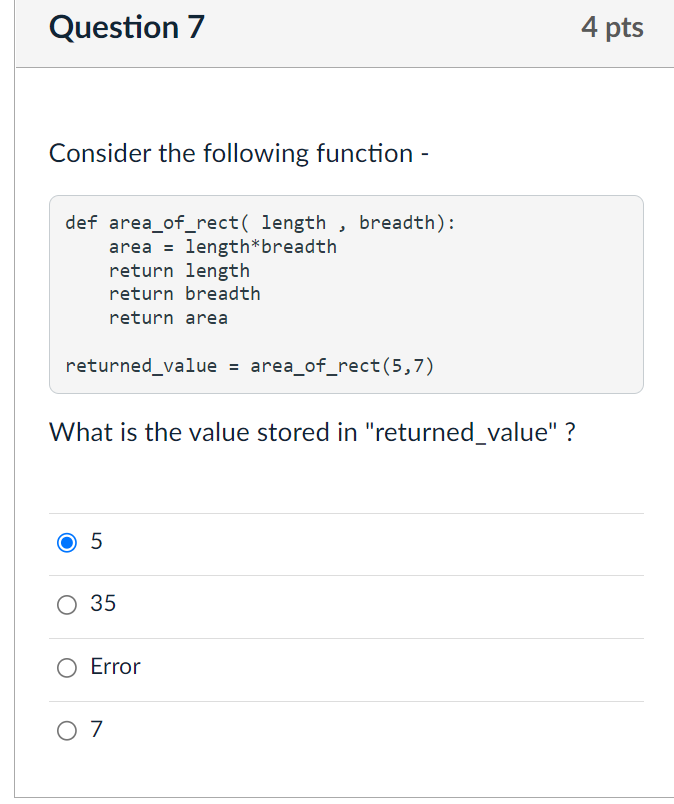


这个例子给args赋值最后是FALSE是赋给args，如果想给option赋值，这时必须key argument to it like option=False.分

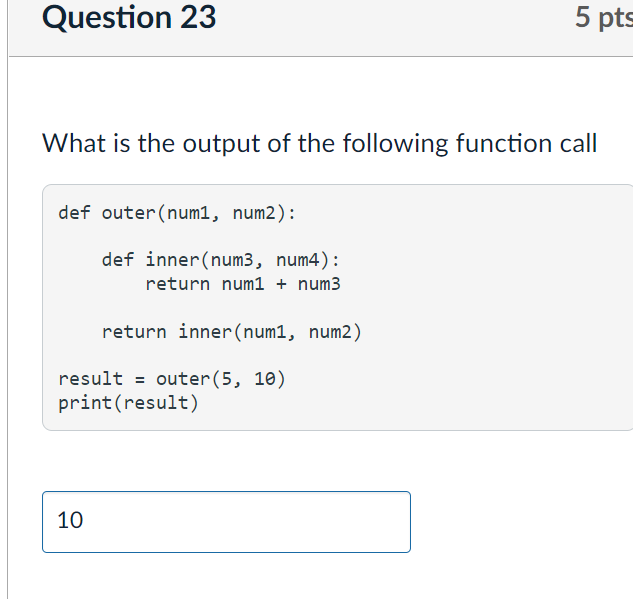
## 5.5 anonymous function



quiz:



第一个return就退出函数了，后两个return就没有运行。



# 6 tuple and string

## 6.1 string index

str1.index('LLO', 2, 6)从位置2开始到位置5之间找，不包括6

str1.find('lLO'，2)从位置2开始到结束之间找

rindex，rfind从尾部开始找

find will return -1, if it can’t find the substring.

要想获取多个字符串的位置，只能自己编函数。

str1[-3: ]-> LD!

str1[-6:-1]-> WORLD

txt.split() default is space

lst2 = txt.split(", ", 2), maxsplit number is 2

### Joining an iterable into one String

"#".join(list1) or “%”.join(string) join(list) join(tuple) join(set), will be

ok, however, join(set) can’t guarantee the order because set is unordered

listNumbers[0:-1]#get all the elements except the last one

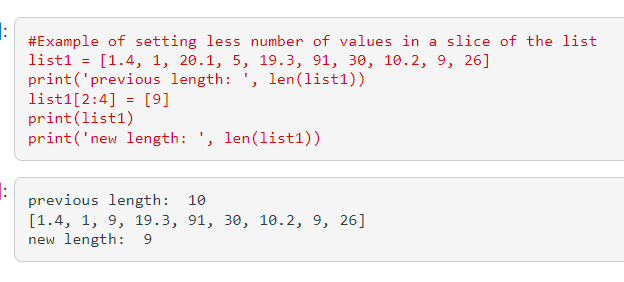
listNumbers[-3:] #get the last three elements

listNumbers[::-1] #reverse the list

listNumbers[10:0:-1] #reverse the list at the specific index

If you insert more elements than you replace, the new elements will be inserted where you specified, and the remaining elements will move accordingly. However, the length of the list will change when the number of elements inserted doesn't match the number of elements replaced.

Similarly, if you insert fewer elements than you replace, the new elements will be inserted where you specified, and the remaining elements will move accordingly. Once again, the length of the list will change. pay attention：both 20.1 and 5 are replaced by 9



insert(index, element)

colors.insert(3,'green') insert函数只能插入一个值如果想插入多个值，得用切片。

my\_list = my\_list[:2] + [6, 7, 8] + my\_list[2:]

A list has several methods that allow you to change it: append( ), insert( ), sort( )... strings have none of these. You can't change a string once it's created. Of course, you can reassign the variable to a different string:

[*item* for *item* in *iterable*]

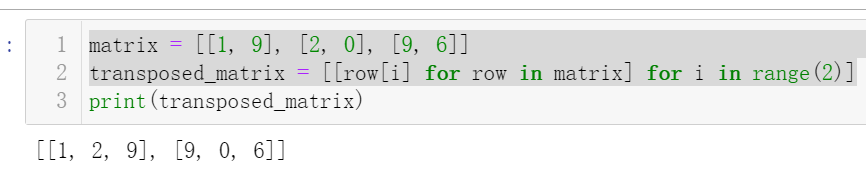
[*item*  for *item* in *iterable* if *condition* == True]

if-else

lst1 = ["even" if i%2 == 0 else "odd" for i in range(10)]

the placement of the "if" statement depends on whether you want to filter the values generated by the loop or conditionally assign values to the list elements.

如果产生的值是for-loop形成的，那么for就在前面，如果产生的值是if形成的，那么if就在前面

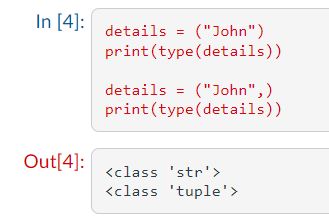


## 6.2 tuple

函数返回多个值，这些值的组合就是tuple.

Tuple elements are ordered, indexed, unchangeable, and allow duplicate values. Tuple elements can be of any data.

If you want to have a tuple with only one element.



the [:-1] slice specifies that we want to include all elements of the list from the beginning up to, but not including, the last element.

 tuple('abc') returns ('a', 'b', 'c') and tuple( [1, 2, 3] ) returns (1, 2, 3)

but tuple({'abc'}) returns(‘abc’,)

（“abc”，）带逗号才能表示一元元组。

拆包：

t = (1,2,3,4)

a,b= t[::2]

output: a=1，b=3

连接：

t1 = (1,2);t2 = (3,4)

t3 = t1+t2

t3 = t3[:2]+(10,)+t3[2:]

output: (1, 2, 10, 3, 4)

## 6.3 set

在Python中，集合（set）是一种可变的无序集合，它是由一组互不相同的元素组成的，具有去重的特性。集合是一种非常实用的数据类型，在处理数据时能够快速实现去重、判断元素是否存在等功能。下面介绍一下set的增删改查操作。

增加元素：

可以通过add()方法向set中添加单个元素，也可以通过update()方法向set中添加多个元素。

pythonCopy code

# 创建一个空集合

my\_set = set()

# 使用 add() 方法添加单个元素

my\_set.add(1)

# 使用 update() 方法添加多个元素

my\_set.update([2, 3, 4])

print(my\_set) # 输出结果：{1, 2, 3, 4}

删除元素：

可以通过remove()方法删除set中的元素。如果要删除的元素不存在，会抛出KeyError异常。还可以通过discard()方法删除元素，如果元素不存在，不会抛出异常。

pythonCopy code

my\_set = {1, 2, 3, 4}

# 使用 remove() 方法删除元素

my\_set.remove(3)

print(my\_set) # 输出结果：{1, 2, 4}

# 使用 discard() 方法删除元素

my\_set.discard(5) # 不会抛出异常print(my\_set) # 输出结果：{1, 2, 4}

修改元素：

set中的元素是不可变的，因此无法修改单个元素。如果要修改set中的元素，需要先删除原来的元素，再添加新的元素。

pythonCopy code

my\_set = {1, 2, 3, 4}

# 先删除原来的元素，再添加新的元素

my\_set.remove(3)

my\_set.add(5)

print(my\_set) # 输出结果：{1, 2, 4, 5}

查询元素：

可以通过in关键字来判断一个元素是否在set中存在。

python Copy code

my\_set = {1, 2, 3, 4}

if 3 in my\_set:

print('3存在于集合中')

else:

print('3不存在于集合中')

| (union operator): The union operator is used to combine two sets into a single set that contains all the elements of both sets, without duplicates. For example, if A = {1, 2, 3} and B = {2, 3, 4}, then A | B = {1, 2, 3, 4}.

& (intersection operator): The intersection operator is used to find the common elements between two sets. In other words, it returns a set that contains only the elements that are present in both sets. For example, if A = {1, 2, 3} and B = {2, 3, 4}, then A & B = {2, 3}.

(difference operator): The difference operator is used to find the elements that are in one set but not in another. For example, if A = {1, 2, 3} and B = {2, 3, 4}, then A - B = {1} and B - A = {4}.

^ (symmetric difference operator): The symmetric difference operator is used to find the elements that are in either of the sets, but not in both. In other words, it returns a set that contains only the elements that are present in either A or B, but not in both. For example, if A = {1, 2, 3} and B = {2, 3, 4}, then A ^ B = {1, 4}.

(proper subset operator): The proper subset operator is used to determine whether one set is a proper subset of another. In other words, it returns true if all the elements of set A are also present in set B, and B contains at least one element that is not in A. For example, if A = {1, 2} and B = {1, 2, 3}, then A > B is false, but B > A is true.

< (proper superset operator): The proper superset operator is used to determine whether one set is a proper superset of another. In other words, it returns true if all the elements of set B are also present in set A, and A contains at least one element that is not in B. For example, if A = {1, 2} and B = {1, 2, 3}, then A < B is true, but B < A is false.

## 6.4 list comprehension

matrix = [[1, 9], [2, 0], [9, 6]]

transposed\_matrix = [[row[i] for row in matrix] for i in range(2)]

print(transposed\_matrix)

#先执行 for i in range(2)在执行括号里的

output： [[1, 2, 9], [9, 0, 6]]

transposed\_matrix = [row[i] for row in matrix for i in range(2)]

print(transposed\_matrix)

#先执行前面的for，后执行后面的for

output：[1, 9, 2, 0, 9, 6]

transposed\_matrix = [row[i] for i in range(2) for row in matrix]

print(transposed\_matrix)

#先执行前面的for，后执行后面的for

output：[1, 2, 9, 9, 0, 6]

# Module C

### Class Variables/Attributes 每个instance都一样的attribute

class Shark:

animal\_type = "fish"

location = "ocean"

### Instance Variables/Attributes每个instance有自己的attribute

class Shark:

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

self.name = name

self.age = age

# 8. inheritance

## 8.1 inheritance

“is a relationship”

class A:

class B(A):

子类不继承\_\_init\_\_,将会覆盖父类的\_\_init\_\_

继承的话就是：

class A:

def \_\_init\_\_(self, fname, lname):

class B(A):

def \_\_init\_\_(self, fname, lname):

A.\_\_init\_\_(self, fname, lname)

super().\_\_init\_\_(fname,lname)

improve properties and methods

def \_\_init\_\_(self, fname, lname, year):

super().\_\_init\_\_(fname, lname)

self.graduation\_year = year

def welcome(self):

print("Welcome")

只继承父类的某些attribute，比如父类的age我不想继承。

class Person:

def \_\_init\_\_(self, fname, lname, age):

self.fname = fname

self.lname = lname

self.age = age

class Student(Person):

def \_\_init\_\_(self, fname, lname, major):

super().\_\_init\_\_(fname, lname, age=None)

self.major = major

self.age = 20

## 8.2 Multiple Inheritance

class Airplane(Vehicle,FlyingVehicle):

def \_\_init\_\_(self, speed, doors, passengers, capacity, propulsion\_mechanism, engines, air\_type):

Vehicle.\_\_init\_\_(self, speed,doors, passengers, capacity)

FlyingVehicle.\_\_init\_\_(self, propulsion\_mechanism,engines)

self.air\_type=air\_type

self.passengers=passengers

self.capacity=capacity

def airborne(self):

FlyingVehicle.fly(self) #在自己的method中调用父类的方法

子类的init该有什么properties就声明什么，在def里定义继承哪个父类或者自己定义

8.3 MRO

print(MultiDerived.\_\_mro\_\_)，返回元祖

print(MultiDerived.mro())，返回列表

我们将 Animal 类放在 Friend 类之前，以确保在调用 super().\_\_init\_\_() 时优先调用 Animal 类的构造函数。然后我们再手动调用 Friend 类的构造函数，以初始化从 Friend 类继承的属性。

如果 Dog 类同时继承了 Animal 和 Friend 两个类，并且这两个类都有 \_\_init\_\_() 方法，那么在子类的构造函数中，调用 super().\_\_init\_\_() 会调用其中一个父类的构造函数，具体是哪个父类的构造函数取决于 Dog 类中的继承顺序（MRO），而不是默认调用第一个父类的构造函数。

需要注意的是，如果多个父类中都有同名的方法，那么 Python 会按照继承顺序调用其中的一个方法，并且调用时的参数也会按照继承顺序传递。因此，在设计多继承的类时，需要注意避免方法名和参数名的冲突。

class X:

def rk(self):

print(" In class X")

class Y(X):

def rk(self):

print(" In class Y")

class Z(X):

def rk(self):

print(" In class Z")

class M(Y,Z):

def \_\_init\_\_(self):

print("Constructor M")

[<class '\_\_main\_\_.M'>, <class '\_\_main\_\_.Y'>, <class '\_\_main\_\_.Z'>, <class '\_\_main\_\_.X'>, <class 'object'>]

对于protected， A.\_protectedAttribute调用，或者如果有get\_protectedAttribute（）的方法可以直接调用。

对于private， A.\_\_privateAttribute是调用不了的，可以用get\_privateAttribute（）的方法调用，

## 8.4 Composition

class Salary:

*#salary class constructor*

def \_\_init\_\_(self, pay, bonus):

self.pay = pay

self.bonus = bonus

*#salary class method*

def annual\_salary(self):

return (self.pay \* 12) + self.bonus

class Employee:

*#employee class constructor*

def \_\_init\_\_(self, name, age, pay, bonus):

self.name = name

self.age = age

*#creating object of salary class*

self.obj\_salary = Salary(pay, bonus) *#Composition 把另一个类的instance作为自己的attribute*

*#employee class method*

def total\_salary(self):

*#calling annual\_salary() method of salary class*

return self.obj\_salary.annual\_salary() 当创建employee类的instance是可以从Salary的instance调取相应方法。

父类不存在，子类就不存在的意思是，如果父类不存在，子类没有实际的物理意义。并不是说子类的代码就有问题，或者说代码就不存在了。

composition 是指在一个类中创建另一个类的实例，将其作为自己的成员变量，并在该类中调用该对象的方法来实现功能。这种关系通常称为“is-a kind of”关系。

使用 composition 的好处是，它允许我们创建更复杂的对象和类之间的关系，同时也提高了代码的可维护性和可重用性。如果我们想要更改 Salary类的实现，或者想要在 Employee类中添加其他成员变量，我们只需要修改对应的类即可，而不需要修改整个程序。

Composition 中的两个对象之间是具有独立性的，一个对象的变化并不会影响到另一个对象，因此，Composition 也称为“弱关联”。

8.5 Aggregation

Aggregation represents "has a" relationship

在 aggregation 中，如果没有特殊说明，你可能无法知道类的 \_\_init\_\_ 方法中的参数是一个 attribute 还是一个 aggregation。

class Course:

def \_\_init\_\_(self, name, code):

self.name = name

self.code = code

def course\_score(self):

pass

class Student:

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

self.name = name

self.age = age

self.course = course

def course\_score():

return course.course\_score（）

python\_course = Course("Python", "CSCI101")

alice = Student("Alice", 20, pythonc\_ourse)

Composition 和 Aggregation 都是面向对象编程中用于描述对象之间关系的概念，它们的异同点如下：

相同点：

都是用于描述类之间的关系。

都可以用于将一个类的对象作为另一个类的对象的成员。

不同点：

Composition 表示一个对象包含另一个对象，被包含的对象是该对象的组成部分，被包含的对象生命周期和包含对象的生命周期一致。Composition 表示一种强依赖关系，被包含的对象只能属于一个包含对象。在代码实现上，通常通过将一个类的实例作为另一个类的成员变量来实现。

Aggregation 表示一个对象包含另一个对象，被包含的对象是该对象的一部分，但是被包含的对象的生命周期可以独立于包含对象的生命周期。Aggregation 表示一种弱依赖关系，被包含的对象可以属于多个包含对象，也可以在不被任何对象包含的情况下存在。在代码实现上，通常通过将一个类的实例作为另一个类的参数传入方法中来实现。

在使用时，可以根据实际情况选择使用 Composition 或 Aggregation。一般来说，如果一个对象是另一个对象的组成部分，且只属于该对象，那么可以使用 Composition；如果一个对象是另一个对象的一部分，但是可以独立存在，或者可以属于多个对象，那么可以使用 Aggregation。需要注意的是，在实际开发中，Composition 和 Aggregation 可能会产生混淆，因此需要根据实际情况进行判断和选择。

本质来说，就是Composition 是在类内建立instance，Aggregation是在类外建立instance。

## aggregation 用setter去配置

## aggregation

## class Building:

## def \_\_init\_\_(self, building\_id, name, location):

## self.\_\_building\_id = building\_id

## self.\_\_name = name

## self.\_\_location = location

## self.\_\_manager = None #在init里我们可以不定义一个aggregation，而是去 #setter去配置

## def get\_manager(self):

## return self.\_\_manager

## 

## def set\_manager(self,manager):

## if not isinstance(manager,Manager):

## raise ValueError('the argument must be a Manager object')

## self.\_\_manager = manager

## class Manager:

## def \_\_init\_\_(self, name, emp\_id, contact\_no):

## self.\_\_name = name

## self.\_\_emp\_id = emp\_id

## self.\_\_contact\_no = contact\_no

Composition 用setter配置

class ExpensiveBuilding(Building):

def \_\_init\_\_(self, building\_id, name, location, amount, authorised\_by):

super().\_\_init\_\_(building\_id, name, location)

self.\_\_amount = amount

self.\_\_authorised\_by = authorised\_by

self.\_\_building\_expense= BuildingExpense(building\_id, amount, authorised\_by)

def get\_building\_expense(self):

return self.\_\_building\_expense

def set\_building\_expense(self, building\_expense):

if not isinstance(building\_expense, Building\_expense):

raise ValueError('building\_expense must be a Building\_expense object')

self.\_\_building\_expense = building\_expense

class BuildingExpense:

def \_\_init\_\_(self, building\_id, amount, authorised\_by):

self.\_\_amount = amount

## polymorphism

私有变量不应该在类的外部直接访问，而应该在类内部访问。因此，通过在类中定义一个方法来访问和修改私有变量是很常见的做法。

使用 @property 装饰器可以将方法转换为属性，从而使属性的访问方式更像属性而不是方法。而使用 @radius.setter 装饰器可以将方法转换为属性的写方法，这样就可以方便地修改私有变量的值。这样做的好处是，外部代码可以像访问普通的属性一样访问和修改私有变量，而不需要知道内部实现的细节，从而简化了代码的使用和维护。

class MyClass:

\_my\_var = None

password = "my\_password" # 设定类属性 password

@property

def my\_var(self):

if self.password == input("Please enter the password: "): # 在返回私有变量前要求输入密码

return self.\_my\_var

else:

print("Incorrect password!")

@my\_var.setter

def my\_var(self, value):

if self.password == input("Please enter the password: "): # 在设置私有变量前要求输入密码

self.\_my\_var = value

else:

print("Incorrect password!")

@property #可以单独出现

def my\_var(self):

@my\_var.setter #这个setter必须在有前面的@property才可以有。

def my\_var(self, value):

@classmethod是Python中的一个装饰器(decorator)，用于指示一个方法是一个类方法(class method)。

类方法是与类相关联的方法，而不是与实例相关联的方法。类方法的第一个参数是类本身(cls)，而不是实例(self)，

@staticmethod 是 Python 中的一个装饰器(decorator)，用于指示一个方法是一个静态方法(static method)。

静态方法是与类相关联的方法，但不需要访问类或实例的任何属性。静态方法的定义中不包含self或cls参数，因此无法访问类的实例变量或类变量。它只能访问在方法定义时作为参数传递的参数以及在方法内部定义的局部变量.

静态方法是一种特殊类型的方法，它不依赖于类实例进行调用。静态方法使用 @staticmethod 装饰器定义。静态方法可以直接通过类名进行调用，而无需创建类实例。静态方法与类的状态无关，它们通常用于执行一些与类相关但不需要访问类属性或实例属性的操作。

以下是一些情况下，我们可以使用静态方法：

当我们需要在类中定义一个独立于任何实例的方法时，可以使用静态方法。

当我们需要在类中定义一个方法，但该方法与实例属性无关时，可以使用静态方法。

当我们需要在类中定义一个方法，但不想让该方法访问类的状态时，可以使用静态方法。

也就是说下面的例子中静态方法isAdult(age)中的age跟类里面的age属性是没有关系的，需要单独传递数据给isAdult(age)中的age。

from datetime import date

class Person:

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

self.name = name

self.age = age

# class method (create a Person object by birth year)

@classmethod

def fromBirthYear(cls, name, year):

return cls(name, date.today().year - year)

# static method (check if a Person is adult or not)

@staticmethod

def isAdult(age):

return age > 18

p1 = Person('john', 26)

p2 = Person.fromBirthYear('john', 1996)

print(p1.age)

print(p2.age)

print(Person.isAdult(19))#直接用类名字调用，无需实例调用，当然用实例也是可以调用的

封装的使用场景：

将类的内部实现细节隐藏起来，避免外部代码不必要地依赖类的实现细节，从而提高代码的可维护性和可重用性。

保护类的属性和方法，避免外部代码直接访问或修改属性和方法，从而提高代码的安全性。

对外暴露必要的接口，方便外部代码使用类的功能。

常见的封装案例：

将属性私有化，并提供公共的get和set方法来访问和修改属性。

将方法私有化，并提供公共的调用方法来调用方法。

使用访问限制（如@property、@classmethod、@staticmethod等装饰器）来限制属性和方法的访问权限。

使用属性描述符（如\_\_getattribute\_\_、getattr、\_\_setattr\_\_等）来控制属性的访问和修改。

使用封装来实现接口隔离、单一职责原则等设计模式。

class MyClass:

def \_\_init\_\_(self, value):

self.\_\_value = value

def get\_value(self):

return self.\_\_value

def set\_value(self, value):

self.\_\_value = value

def \_\_my\_private\_method(self):

print("This is a private method")

def my\_public\_method(self):

self.\_\_my\_private\_method()

# 创建一个MyClass的实例对象

obj = MyClass(42)

# 调用my\_public\_method方法，间接地调用\_\_my\_private\_method方法

obj.my\_public\_method() # 输出：This is a private method

类变量 可用来计数，每创建一个实例count加1，

另外类常量可以作为全局变量使用。具体的问chatgpt

class MyClass:

count = 0

def \_\_init\_\_(self):

MyClass.count += 1

return cls().name 和 return cls.name 是两个不同的语句，含义也不同。

return cls().name 创建一个类的实例，并返回该实例的 name 属性值。这里，cls() 创建一个类的实例，name 是存储 name 属性值的实例变量。当我们想要访问类的实例变量时，这种方法很有用。

另一方面，return cls.name 返回类变量 name 的值。类变量是在类定义中定义的，并且由类的所有实例共享。当我们想要访问类变量时，这种方法很有用