>>> x = object() >>> X <object object at 0x7ff2e7b0d0f0> object是所有类的基类,他带有所有Python类实 >>> type(x) 例均带有的通用方法。 <class 'object'> >>> isinstance(str, object) True class object() 这个函数不接受任何参数。 返回一个不带特征的新对象。 由于object没有\_\_dict\_\_,所以无法将任意属性赋 值给object的实例。 在具有单继承的类层级结构中, super 可用来引 用父类而不必显式地指定它们的名称,从而令代 码更易维护。 这种用法与其他编程语言中 super 的用法非常相似。 此外super还可以在动态环境中支持协作多重继 承。这使用实现"菱形图"成为可能,即有多个基 super(type) 类实现相同的方法。 返回一个代理对象,由它将方法调用委托给type 的父类或兄弟类。 class Father: 这对于访问已在类中被重载的继承方法很有用。 def greetings(self): print("hello from father!") class Son(Father): def greetings(self): class super() super().greetings() 返回父类或兄弟类的代理对象 print("hello from son, too!") >>> s = Son() >>> s.greetings() hello from father! hello from son, too! super(type, object\_or\_type) object\_or\_type 确定要用于搜索的 method resolution order. 搜索会从 type 之后的类开始。 >>> type("hello") class type(object) <class 'str'> 返回给定对象的类型。 >>> type(123) <class 'int'> class type() 返回对象的类型,或者创建一种新的类型。 以下两个语句会创建相同的type对象: class type(name, bases, dict, \*\*kwds) 返回一个新的type对象。 >>> class A: ... v = 1 这本质上是class语句的一种动态形式,name成 为类型、bases成为基类(为空则为终极基类 >>> A = type("A", (), dict(v=1)) object)、dict包含类主体的属性和方法定义。 fget是获取属性值的函数。 fset是用于设置属性值的函数。 fdel是用于删除属性值的函数。 doc为属性对象的文档字符串。 class Parrot: def \_\_init\_\_(self): self.\_voltage = 100000 @property def voltage(self): """Get the current voltage.""" return self.\_voltage @property 装饰器会将 voltage() 方法转化为一 个具有相同名称的只读属性 "getter", 并将 property()可以作为装饰器来实现只读属性。 voltage 的文档字符串设为 "Get the current class property(fget=None, fset=None, voltage." fdel=None, doc=None) 返回 property 属性。 >>> x = Parrot() >>> x.voltage >>> # 无法改写只读属性 >>> x.voltage = 1 Traceback (most recent call last): File "<stdin>", line 1, in <module> AttributeError: can't set attribute 'voltage' class C: def \_\_init\_\_(self):  $self_x = None$ @property def x(self): """I'm the 'x' property.""" return self.\_x @x.setter def x(self, value):  $self_x = value$ 特征属性对象具有 getter, setter 和 deleter 方 法,它们可用作装饰器来创建该特征属性的副 @x.deleter def x(self): 本,并将相应的访问函数设为所装饰的函数。 del self.\_x 上述代码与前一个例子完全等价。 注意一定要给附加函数与原始的特征属性相同的 名称 (在本例中为 x。) >>> bool() False >>> bool(1) True >>> bool(0) class bool(x=False) 根据x的值为真或者假,返回布尔值True或者 False 返回一个布尔值对象。 >>> bool("") False. False >>> bool("hi") True >>> bool(True) True >>> float(3.14) >>> float("3.14") class float(x=0.0) >>> float("+3.14") 类构造器 根据给定的数字或字符串生成浮点数 返回一个浮点数对象 >>> float("314e-002") >>> float("-Infinity") >>> int() >>> int(10086) 返回一个基于数字或字符串 x 构造的整数对象, 或者在未给出参数时返回 0。 >>> int(10086.11) 对于浮点数,这将向零方向截断。 10086 >>> int("10086") class int(x=0)class int(x, base=10) 返回一个整数对象。 如果 x 不是一个数字或者如果给定了 base,则 x 必须是一个表示以 base 为基数的整数的字符 >>> int("0xFF", 16) 串、bytes 或 bytearray 实例。 字符串前面还能加上可选的 + 或 - (中间没有空 格), 带有前导的零, 带有两侧的空格, 并可带有 数位之间的单个下划线。 >>> complex(3, 4) (3+4j)class complex(real=0, imag=0) 返回值为 real + imag\*1j 的复数,或将字符串或 class complex(string) 数字转换为复数。 返回一个复数对象。 >>> complex("5+8j") (5+8j)>>> str() str(object) >>> str([1, 2, 3]) 返回给定对象的字符串表示。 '[1, 2, 3]' >>> str({'a', 'b', 'c'}) "{'a', 'c', 'b'}" class str() 返回一个字符串对象。 str(object=b", encoding='utf-8', errors='strict') 在给出encoding或是errors至少其中一个时, object应该是一个bytes-like对象(比如bytes或 >>> str(b"hello", encoding='utf-8') bytearray) . 'hello' 在此情况下,如果object是一个bytes或 bytearray对象,那么str(bytes, encoding, errors) 等价于bytes.decode(encoding, errors)。 class dict(\*\*kwarg) class dict(mapping, \*\*kwarg) 创建一个新的字典。 >>> dict(id=10086, name="huangz") class dict(iterable, \*\*kwarg) {'id': 10086, 'name': 'huangz'} dict对象是一个字典类。 返回一个字典对象。 >>> list() class list >>> list("hello") class list(iterable) 构建可变序列。 ['h', 'e', 'l', 'l', 'o'] 返回一个列表对象。 >>> list(range(5)) [0, 1, 2, 3, 4] 返回一个新的bytes数组。 >>> bytearray(b'10010') class bytearray(source=b") class bytearray(source, encoding) bytearray类是一个可变序列,包含范围为 bytearray(b'10010') class bytearray(source, encoding, errors) >>> bytearray(u'你好', encoding='utf-8') 0<=x<=256的整数,其具有可变序列的大部分常 返回一个字节数组对象。 bytearray(b'\xe4\xbd\xa0\xe5\xa5\xbd') 见方法。 class bytes(source=b") 返回一个新的bytes对象,这是一个不可变序列, class bytes(source, encoding) 包含范围为0<=x<=256的整数。 >>> bytes(b'10010') class bytes(source, encoding, errors) b'10010' bytes是bytearray的不可变版本, 带有同样不改 返回一个不可变的字节数组对象,这是bytearray 变序列的方法, 支持同样的索引、切片操作。 的不可变版本。 >>> range(5) range(stop) range(0, 5)创建从0至stop-1为止的数字序列,相当于执行 >>> list(range(5)) range(0, stop)或range(0, stop, 1)。 [0, 1, 2, 3, 4] >>> range(1, 5)range(start, stop) range(1, 5) >>> list(range(1, 5)) class range() 创建从start至stop-1为止的数字序列,相当于执 创建一个不可变的数字序列。 行range(start, stop, 1)。 [1, 2, 3, 4] >>> range(0, 10, 2) range(0, 10, 2) >>> list(range(0, 10, 2)) range(start, stop, step=1) [0, 2, 4, 6, 8] 创建从start至stop-1为止,步进为step的数字序 >>> # step 还可以是负数 >>> list(range(0, -10, -2)) [0, -2, -4, -6, -8]set() >>> set() 返回一个空白的set对象。 set() class set(iterable=None) 创建一个集合对象。 set(iterable) >>> set([1, 2, 2, 3, 3, 3, 4]) 返回一个set对象,其中包含从iterable获取的元 {1, 2, 3, 4} >>> frozenset() class frozenset(iterable=set()) 返回一个新的frozenset对象,其中包含可选参数 frozenset() 创建一个不可变的集合对象。 >>> frozenset(range(5)) iterable中的元素。 frozenset({0, 1, 2, 3, 4}) slice(stop) 相当于执行slice(None, stop, None) class slice() >>> slice(5) slice(start, stop, step=None) 创建切片对象。 slice(None, 5, None) 创建一个包含指定索引集的切片对象,这种对象 >>> slice(0, 5) 通常只包含特定序列的一部分对象。 slice(0, 5, None) 其中start、stop和step为只读属性,它们除了简 >>> slice(0, 5, None) 单地返回相应的参数值之外没有其他显式功能。 slice(0, 5, None) tuple() >>> tuple() 创建空白元组。 class tuple() tuple(iterable) >>> tuple(range(3)) 创建不可变的元组序列。 根据iterable创建与之对应的不可变元组。 (0, 1, 2)元组实现了所有一般序列的操作。 class memoryview(object) 返回由给定实参创建的"内存视图"对象。 返回由给定对象创建的内存试图对象。 object可调用时返回True, 否则返回False。 >>> callable(abs) 返回True不保证调用一定成功,但返回False则肯 callable(object) 定不会成功。 >>> callable("hi") (类是可调用的,带有\_call\_()方法的类实例也 False 是可调用的。) >>> hash(10086) 10086 返回对象的哈希值,该值为整数。 >>> hash(10086.0) 字典在查找元素时会使用这个函数来快速比较字 10086 hash(object) >>> hash("hi") 相同大小的数字变量拥有相同的哈希值,即使它 6086364370360132798 们的类型不同(比如100和100.0)。 >>> hash("hello world!") 8647857355983603518 返回对象的标识值。 >>> id(10086) 139848762387472 这个值是一个整数,它在对象的生命周期中保证 id(object) 是唯一且恒定的。 >>> x = 10086两个生命周期不重叠的对象可能具有相同的标识 >>> id(x)139848762385168 >>> class O: ... def \_\_init\_\_(self, x, y): self.x = x ... self.y = y >>> o1 = O(1, 2) >>> o1.x 为对象的指定属性赋值。 执行setattr(obj, attr, value)等同于执行 setattr(object, name, value) >>> setattr(o1, 'x', 10086) obj.attr=value。 >>> o1.x 10086 >>> o1.y >>> o1.y = 256 >>> o1.y 256 >>> class O: ... def \_\_init\_\_(self, x, y): = ... self.x = x  $\dots$  self.y = y >>> o1 = O(1, 2) 获取对象指定属性的值。 >>> getattr(o1, 'x') 执行getattr(obj, attr)等同于执行obj.attr。 getattr(object, name) getattr(object, name, default) 如果指定属性不存在,那么返回default,否则引 >>> getattr(o1, 'y') 发AttributeError。 >>> getattr(o1, 'z') Traceback (most recent call last): File "<stdin>", line 1, in <module> AttributeError: 'O' object has no attribute 'z' >>> getattr(o1, 'z', 3) >>> hasattr(str, "join") 检查对象是否具有指定属性,是的话返回True, hasattr(object, name) >>> hasattr(str, "some crazy attr") 否则返回False。 类/对象相关 False >>> class O: ... def \_\_init\_\_(self, x, y): ... self.x = x $\dots$  self.y = y >>> o1 = O(1, 2)>>> o1.x 删除对象的指定属性。 >>> delattr(o1, 'x') delattr(object, name) >>> o1.x 执行delattr(obj, attr)等同于执行del obj.attr。 Traceback (most recent call last): File "<stdin>", line 1, in <module> AttributeError: 'O' object has no attribute 'x' >>> o1.y >>> del o1.y >>> o1.y Traceback (most recent call last): File "<stdin>", line 1, in <module> AttributeError: 'O' object has no attribute 'y' 如果 object 参数是 classinfo 参数的实例,或者 是其 (直接、间接或 虚拟) 子类的实例则返回 >>> s = "hi" 如果 object 不是给定类型的对象,则该函数总是 >>> isinstance(s, str) isinstance(object, clssinfo) 返回 False。 True >>> isinstance(s, list) 如果 classinfo 是由类型对象结成的元组 (或是由 **False** 其他此类元组递归生成)或者是多个类型的 union 类型,则如果 object 是其中任一类型的实 例时将会返回 True。 >>> class AnotherStr(str): 如果 class 是 classinfo 的子类 (直接、间接或 >>> issubclass(AnotherStr, str) 虚的),则返回 True。 类将视为自己的子类。 issubclass(class, classinfo) classinfo 可为类对象的元组(或递归地,其他这 >>> issubclass(str, str) 样的元组)或 union 类型,这时如果 class 是 True classinfo 中任何条目的子类,则返回 True。 >>> issubclass(str, list) False 把方法封装为类方法。 >>> class C: ... @classmethod - 类方法隐含的第一个参数就是类,正如实例方 ... def f(cls): 法接收实例作为第一个参数一样。 @classmethod ... print("hello world!") - 类方法的调用可以在类上进行(比如c.f())也 可以在实例上进行(比如c().f())。 >>> C.f() - 类方法不同于C++或JAVA中的静态方法,需要 hello world! 后者的话应使用staticmethod()。 使用装饰器语法定义静态方法: class C: @staticmethod def f(arg1, arg2, argN): @staticmethod 将方法转换为静态方法。 使用常规函数调用语法定义静态方法: def regular\_function(): pass class C: method = staticmethod(regular\_function) 返回一个iterator对象。 >>> iter("hello") 单参数调用的情况下, object必须支持iterable协 <str\_iterator object at 0x7f3113e53520> 议 (带有\_iter\_()方法) 或者序列协议 (带有 >>> list(iter("hello")) \_\_getitem\_\_()方法)。 ['h', 'e', 'l', 'l', 'o'] 不支持这些协议将引发TypeError。 iter(object) iter(object, sentinel) 给出第二个参数sentinel时,则object必须是一个 从二进制数据库文件中读取固定宽度的块,直至 可调用对象。 到达文件的末尾: 在这种情况下创建的迭代器将持续地不带参数调 from functools import partial 用object, 直到其返回值等于sentinel并引发 with open('mydata.db', 'rb') as f: for block in iter(partial(f.read, 64), b"): StopIteration为止, 否则就返回object返回的 process\_block(block) 通过调用iterator的\_next\_()方法获得下一个元 >>> next(iter([1, 2, 3])) next(iterator) next(iterator, default) 如果迭代器耗尽,那么返回给定的default,没有 >>> next(iter([]), 10086) 10086 给定default则触发StopIteration。 返回一个枚举对象。 >>> fruits = ["Apple", "Banana", "Cherry"] >>> list(enumerate(fruits)) iterable必须是序列、iterator或其他支持迭代的 [(0, 'Apple'), (1, 'Banana'), (2, 'Cherry')] enumerate(iterable, start=0) enumerate()返回的迭代器的\_\_next\_\_()方法返回 >>> list(enumerate(fruits, start=1)) 一个元组,里面包含一个计数值(从start开始, [(1, 'Apple'), (2, 'Banana'), (3, 'Cherry')] 默认为0) 和通过迭代iterable获得的值。 >>> len("hello") 返回对象的长度(其包含的元素个数)。 len(object) >>> len([1, 2, 3]) 实参可以时序列或是集合。 >>> any([]) **False** iterable的任一元素为真时返回True。 >>> any([False, True]) any(iterable) 若可迭代对象为空则返回False。 True >>> any([False, False]) False >>> all([]) iterable所有值为真 >>> all([True, True]) 或可迭代对象为空时 all(iterable) 返回True >>> all([False, True]) >>> max([3, 1, 2]) 返回可迭代对象中最大的元素,或者两个及以上 实参中最大的一个。 >>> max(3, 1, 2) >>> max([], default=10086) 10086 default用于在iterable为空时用作默认返回值。 >>> max([]) 如果iterable为空并且没有给定这个参数,那么触 Traceback (most recent call last): 发ValueError。 File "<stdin>", line 1, in <module> max(iterable, \*, key=None) ValueError: max() arg is an empty sequence max(iterable, \*, default, key=None) max(arg1, arg2, \*args, key=None) >>> max(['A', 'a']) key参数用于指定排序时用于处理单个项的函 >>> max(['A', 'a'], key=str.lower) 数,它只接受单个输入 >>> str.lower('A') 如果有多个最大的项, 那么返回第一个找到的。 >>> min(3, 1, 2) 返回可迭代对象中最小的元素,或者返回两个及 min(iterable, \*, key=None) min(iterable, \*, default, key=None) 以上实参中最小的一个。 >>> min([3, 1, 2]) min(arg1, arg2, \*args, key=None) 其他参数及其意义跟max()相同。 sum(iterable) >>> sum([1, 3, 5, 7]) 从左向右对iterable的项求和并返回总计算值。 sum(iterable, /, start=0) 在我的Python3.10.12版本上,在给定start参数的 sum(iterable, start) 迭代器相关 情况下, sum()未能正常工作, 有bug? 从索引start上的项开始,对iterable的项进行求 >>> sum([1, 3, 5, 7], start=1) filter(function, iterable) 使用iterable中function返回真值的元素构造一个 >>> list(filter(lambda i: i%2==0, range(10))) iterable可以是序列、支持序列的容器或是迭代 [0, 2, 4, 6, 8] 相当于执行(item for item in iterable if function(item))。 filter(None, iterable) 使用标识号函数,将iterable中所有具有假值得元 >>> list(filter(None, ["hi","",True,False,0,1])) filter python" ['hi', True, 1] 相当于执行(item from item in iterable if item)。 Python 内置函数脑图 >>> import itertools 使用itertools.filterfalse(predicate, iterable)可以 >>> list(itertools.filterfalse(lambda i: i%2==0, ver. 3.12 range(10))) 只返回iterable中对predicate只返回假的元素。 [1, 3, 5, 7, 9] 制作: 黄健宏 >>> list(range(5)) [0, 1, 2, 3, 4] 返回一个迭代器,里面包含了将function应用于 >>> # 让序列数值+1 iterable每一项之后得到的结果。 >>> list(map(lambda x: x+1, range(5))) 如果iterable有不止一个,那么function必须能够 map(function, iterable) [1, 2, 3, 4, 5] map(function, iterable, \*iterable) 同时接受指定数量的iterable。 >>> # 让两个序列相加 在传入多个iterable的时候, 最短的iterable耗尽 >>> list(map(lambda x, y: x+y, range(5), 时整个迭代终止。 range(5))) [0, 2, 4, 6, 8] zip(\*iterable) >>> list(zip(range(3), ["apple", "banana", 在多个迭代器上并行迭代,从每个迭代器返回一 "cherry"])) 个数据项组成元组,其中第i个元组包含的是每个 [(0, 'apple'), (1, 'banana'), (2, 'cherry')] 迭代器的第i个元素。 zip() >>> list(zip(range(3), ["apple", "banana", "cherry"],strict=True)) zip(\*iterable, strict=False) [(0, 'apple'), (1, 'banana'), (2, 'cherry')] 在默认情况下, zip()在最短的迭代完成后停止, 较长可迭代对象中剩余的项将被忽略。 >>> list(zip(range(5), ["apple", "banana", 但如果strict为True,则输入的可迭代对象必须为 "cherry"],strict=True)) 等长,否则就会引发ValueError。 Traceback (most recent call last): File "<stdin>", line 1, in <module> ValueError: zip() argument 2 is shorter than argument 1 >>> import random >>> lst = list(range(5)) >>> random.shuffle(lst) 根据iterable中的项返回一个新的已排序列表。 >>> lst [1, 4, 0, 3, 2] >>> sorted(lst) [0, 1, 2, 3, 4] 具有key和reverse两个可选参数,它们都必须指 定为关键字参数: - key 指定带有单个参数的函数,用于从iterable >>> sorted(lst, reverse=True) sorted(iterable, /, \*, key=None, reverse=False) 的每个元素中提取用于比较的键。默认值为 [4, 3, 2, 1, 0] - reverse为布尔值。如果为True,则对iterable进 行反向排序。 sorted是稳定的,它在排序的时候不会改变比较 结果相等的元素的相对顺序 (这有利于进行多重 排序)。 >>> reversed(range(5)) 返回一个反向的iterator。 <range\_iterator object at 0x7ff2e64dbd20> reversed(seq) seq必须是一个具有\_reversed\_()方法或是支持 >>> list(reversed(range(5))) 序列协议的对象。 [4, 3, 2, 1, 0] >>> abs(-100) 返回整数、浮点数等对象的绝对值 100 abs(x) >>> c = complex(3,4)返回复数的模 >>> abs(c) 5.0 以两个(非复数)数字为参数,在作整数除法 >>> divmod(10, 3) 时,返回商和余数。 (3, 1)对于整数而言,结果与 (a // b, a % b) 相同。 对于浮点数则结果为 (q, a % b), 其中 q 通常为 math.floor(a / b), 但可能比它小 1。 divmod(a, b) 在任何情况下, q\*b+a%b都非常接近a, 如 果 a % b 非零,则结果符号与 b 相同,并且 0  $\neq$  abs(a % b)  $\neq$  abs(b). 若操作数为混合类型,则适用二进制算术运算符 的规则。 >>> 3\*\*5 返回base的exp次幂,效果等价于乘方运算符: 243 base\*\*exp。 >>> pow(3, 5) pow(base, exp, mod=None) >>> pow(3, 5, 7) 如果给定了mod,则返回base的exp次幂对mod 取余的结果 (效率比pow(base, exp)%mod要 数值计算/转换 >>> 3\*\*5%7 round(number) >>> round(3.1415926) 返回最接近输入值的整数。 round(number, ndigits=None) round(number, ndigits) >>> round(3.1415926, 2) 返回将输入值舍入至小数点ndigits位之后的值。 3.14 >>> bin(3) '0b11' >>> bin(10086) '0b10011101100110' bin(x) 将整数转换为带"0b"前缀的二进制字符串。 >>> format(3, '#b') '0b11' >>> format(10086, 'b') '10011101100110' >>> f'{10086:#b}' '0b10011101100110' >>> oct(0) '000' 将整数转换为带"0o"前缀的八进制数字字符串。 oct(x) >>> oct(10) '0012' >>> hex(0) 将整数转换为带"0x"前缀的小写十六进制数字字 '0x0' hex(x) >>> hex(10086) 符串。 '0x2766' >>> ord(u"龙") 40857 >>> chr(40857) 返回Unicode码位为整数i的字符。 chr(i) 这是ord()的逆函数。 >>> chr(ord(u"龙")) 字符编码/解码 >>> ord(u"龙") 返回Unicode字符c的码点整数。 40857 ord(c) 这是chr()的逆函数。 >>> ord('a') 97 >>> ascii("你好") 以字符串方式返回对象的可打印表示,跟repr() "'\\u4f60\\u597d" ascii(object) 类似,但是会使用 \x、\u 和 \U 对 repr() 返回的 >>> repr("你好") 字符串中非 ASCII 编码的字符进行转义。 "'你好'" >>> repr("hello") "'hello'" >>> repr([1, 2, 3]) repr(object) 返回对象的可打印形式字符串表示。 '[1, 2, 3]' >>> repr(str) "<class 'str'>" >>> format(3.1415926, ".2f") '3.14' - format(value, spec=") 根据spec指定的格式对value实施格式化。 更多例子 ď https://docs.python.org/zhcn/3/library/string.html#formatspec >>> print("hello world!") hello world! 将objects打印输出至file指定的文件流,以sep分 隔并在末尾加上end。 >>> print("hello","world","!",sep="+",end="\n") sep、end、file和flush必须以关键字参数的形式 给出, 否则它们就会被视作objects的一部分并被 转换为字符串。 sep和end必须为字符串,或者是代表使用默认值 的None。 print(\*objects, sep=' ', end='\n', file=None, 如果没有给出objects, 那么将只写入end。 flush=False) file必须是一个具有write(string)方法的对象,如 果参数不存在或者为None,则使用sys.stdout。 由于要打印的参数会被转换为文本字符串, 所以 print()不能用于二进制模式的文件对象,这些对 象应该使用file.write()。 打印/输入输出 输出缓冲区通常由file确定, 但如果flush为真 值,那么流将被强制刷新。 input() >>> input() 从输入中读取一行,将其转换为字符串并返回 hello world! (不带末尾的换行符)。 'hello world!' 当读取到EOF时触发EOFError。 - input input(prompt) >>> input("Enter your words:") 如果给定了prompt则将其写入标准输入,末尾不一 Enter your words:hello world! 'hello world!' 带换行符。 >>> f = open("greetings.txt") 打开file并返回对应的file object,如果无法打开 >>> f.read() 'hello world!\n' 则引发OSError。 >>> f.close() file是一个path-like object,表示将要打开的文 件之路径 (绝对路径或当前工作目录的相对路 径),也可以是要封装文件对应的整数类型文件 描述符。 当使用缓冲以二进制模式打开文件时,返回的类 以二进制模式打开的文件,返回的内容为bytes对 是 io.BufferedlOBase 的一个子类。 象, 无需进行编码。 当禁用缓冲时,则会返回原始流,即 io.RawlOBase 的一个子类 io.FileIO。 在文本模式下,文件内容返回为str,并使用指定 open()在以文本模式打开时返回 io.TextlOBase 的encoding或平台默认的字节编码实施解码。 (具体为 io.TextlOWrapper) 的一个子类。 'r' 读取 (默认) mode是一个可选的字符串参数,用于指定文件 'w' 写入,并先截断文件 打开的模式。 'x' 排它性创建,如果文件已存在则失败 'a' 打开文件用于写入,如果文件存在则在末尾追 'b' 二进制模式 't' 文本模式 (默认) '+' 打开用于更新 (读取与写入) open(file, mode='r', buffering=- 1, encoding=None, errors=None, newline=None, closefd=True, opener=None) 可以同时指定多个模式,比如"wb+" 0 关闭缓冲 (只能在二进制模式下使用) buffering是可选的整数参数,用于设置缓冲策 1行缓冲 >1 使用固定大小的缓冲块,给定值就是该块的字 节大小 encoding可选参数用于指定编码格式,并且只适 用于文本模式。 默认编码格式依赖于具体平台,由 locale.getencoding()决定。 errors可选字符串参数用于指定如何处理编码和 解码错误,并且只适用于文本模式。 newline可选参数用于决定如何解释来自流的换行 符,它可以是None,",'\n','\r'和'\r\n'。 当closefd为False并且file为文件描述符而不是文 在file为文件名的情况下, closefd必须为True, 件名的时候,即便文件被关闭,文件对象底层对 否则将引发错误。 应的文件描述符也会继续保持打开状态。 通过使用参数 (file, flags) 调用 opener 获得 文件对象的基础文件描述符。 opener 必须返回 opener用于指定自定义开启器,它必须是可调用 一个打开的文件描述符(使用 os.open as opener 时与传递 None 的效果相同)。 返回当前本地作用域中的名称列表。 >>> dir() dir() ['\_annotations\_', '\_builtins\_', '\_doc\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_spec\_\_'] 具有\_dir\_()方法的对象,将调用该方法,由它 dir 返回属性组成的列表。 尝试返回对象的有效属性列表。 未提供\_dir\_()方法的对象,函数会尽量从对象 所定义的\_\_dict\_\_属性和其类型对象中收集信 dir(object) >>> dir(str) ['\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', ...] >>> globals() {'\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': None, '\_\_package\_\_': None, '\_\_loader\_\_': <class 返回实现当前模块命名空间的字典。 globals() 对于函数内的代码,这是在定义函数时设置的, '\_frozen\_importlib.BuiltinImporter'>, '\_\_spec\_\_': 无论函数在哪里被调用它都不会发生变化。 None, '\_annotations\_': {}, '\_builtins\_': <module 'builtins' (built-in)>} 变量/环境管理 更新并返回表示当前本地符号表的字典。 >>> locals() {'\_name\_': '\_main\_', '\_doc\_': None, 在函数代码块但不是类代码块中调用locals()时将 \_package\_\_': None, '\_loader\_\_': <class locals() 返回自由变量。 '\_frozen\_importlib.BuiltinImporter'>, '\_\_spec\_\_': 在模块层级上, locals()和globals()是同一个字 None, '\_annotations\_': {}, '\_builtins\_': <module 'builtins' (built-in)>} >>> vars() {'\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': None, vars() \_\_package\_\_': None, '\_\_loader\_\_': <class 不带参数调用时,其行为类似locals()。 '\_frozen\_importlib.BuiltinImporter'>, '\_\_spec\_\_': None, '\_annotations\_': {}, '\_builtins\_': <module 'builtins' (built-in)>} vars() vars(object) >>> vars(int) 返回给定模块、类、实例或任何其他具有 mappingproxy({'\_\_new\_\_': <built-in method \_new\_ of type object at 0x564148392320>, ...) \_\_dict\_\_属性对象的\_\_dict\_\_属性。 \_import\_\_(name, globals=None, locals=None, 由import语句发起调用。 fromlist=(), level=0) 3.10新功能 aiter(async\_iterable) 异步 awaitable anext(async\_iterator) 3.10新功能 awaitable anext(async\_iterator, default) 将 source 编译成代码或 AST 对象。 代码对象可以被 exec() 或 eval() 执行。 compile(source, filename, mode, flags=0, dont\_inherit=False, optimize=- 1) source 可以是常规的字符串、字节字符串,或 者 AST 对象。 编译 eval(expression, globals=None, locals=None) exec(object, globals=None, locals=None, /, \*, closure=None) 调试 breakpoint(\*args, \*\*kws) >>> help() Welcome to Python 3.10's help utility! If this is your first time using Python, you should definitely check out the tutorial on the internet at https://docs.python.org/3.10/tutorial/. Enter the name of any module, keyword, or topic to get help on writing help() Python programs and using Python modules. 如果没有实参,解释器控制台里会启动交互式帮 To quit this help utility and return to the interpreter, just type "quit". 助系统。 To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", type "modules spam". help> 帮助 help >>> help(str) Help on class str in module builtins: class str(object) | str(object=") -> str | str(bytes\_or\_buffer[, encoding[, errors]]) -> str | Create a new string object from the given object. If encoding or I errors is specified, then the object must help(request) expose a data buffer | that will be decoded using the given 如果实参是一个字符串,则在模块、函数、类、 encoding and error handler. 方法、关键字或文档主题中搜索该字符串,并在 | Otherwise, returns the result of 控制台上打印帮助信息。 object.\_\_str\_\_() (if defined) 如果实参是其他任意对象,则会生成该对象的帮 | or repr(object). 助页。 | encoding defaults to sys.getdefaultencoding(). | errors defaults to 'strict'. | Methods defined here: \_add\_(self, value, /) Return self+value. \_contains\_(self, key, /) Return key in self. Presented with xmind