学习C/C++之后，自学Python（3.x），刚开始对这门语言不太熟悉，记点笔记。

**最开始最直观的印象**

1. 定义一个变量时，不用int a，或者char c这样，就直接value = 123，不用声明类型，它的赋值机制与c++不太一样，它其实是先开辟一个空间，存放这个数据后，value相当于是一个引用。

还可以x , y = 4 , 8这样同时赋值。

1. score = int(input("请输入分数:"))

if 90 <= score <= 100 : #注意冒号

print('A') #python靠缩进来判断

elif 80 <= score < 90 : #注意冒号

print('B')

elif 60 <= score < 80 : #注意冒号

print('C')

elif 0 <= score < 60 : #注意冒号

print('D')

else : #注意冒号

print("输入错误")

没有分号，没有分号，没有分号

if 、else 、for 、while等后面有冒号，有冒号，有冒号

新增了elif ，就相当于else if

Python 不用｛｝来区分作用域，靠的是缩进，缩进，缩进

1. 除法 1/2 等于0.5 1//2 等于0

幂方运算 2\*\*5 等于32

1. for(i=0;i<4;i++)

{

......

}

在Python里为

for i in range(0,4) :

......

虽然Python是用c语言写的，但是不同之处还是有很多的，Python更注重实用.....

1. **数据类型**
2. 数字
3. 字符串
4. 元组
5. 列表
6. 字典

字符串、列表、元组都为序列类型的数据,序列就可以进行切片操作

1. **数字**

添加了复数类型，j就相当于i

|  |
| --- |
| 1. >>> c=3.14 2. >>> type(c) 3. <class 'float'> 4. >>> d=3.14j 5. >>> type(d) 6. <class 'complex'> 7. >>> c+d 8. (3.14+3.14j) |

没有double,只有float

**2.字符串**

（1）三重引号

|  |
| --- |
| 1. >>> str1='''tom: 2. hello world! ''' 3. >>> str1 4. 'tom:\n hello world! ' 5. >>> print (str1) 6. tom: 7. hello world! |

三重引号也可以用来多行注释

1. 索引

|  |
| --- |
| 1. >>> str1="abcde" 2. >>> str1[0] 3. 'a' 4. >>> str1[4] 5. 'e' 6. >>> str1[5] 7. Traceback (most recent call last): 8. File "<pyshell#3>", line 1, in <module> 9. str1[5] 10. IndexError: string index out of range 11. >>> str1[-1] #-1表示倒数第一个 12. 'e' 13. >>> str1[-2] 14. 'd' 15. >>> str1[0]+str1[-1] 16. 'ae' |

（3）切片操作

|  |
| --- |
| 1. >>> str1[1:4] #序列类型均有的切片操作   'bcd' |

1. 相关方法

|  |  |
| --- | --- |
| 操作 | 含义 |
| + | 连接 |
| \* | 重复 |
| <string>[] | 索引 |
| <string>[ : ] | 切片 |
| len(<string>) | 求长度 |
| <string>.upper() | 将字符串中所有字母大写(但原字符串没有改变) |
| <string>.lower() | 将字符串中所有字母小写 |
| <string>.strip() | 去两边空格及去指定字符 |
| <string>.split() | 按指定字符分割字符串为数组 |
| <string>.join() | 连接两个字符串序列 |
| <string>.find() | 搜索指定字符串 |
| <string>.replace() | 字符串替换 |
| for <var> in <string> | 字符串迭代 |

1. **元组（tuple）**

|  |
| --- |
| >>> t=123,  >>> type(t)  <class 'tuple'>  >>> t=123,456,'abc' #t=（123,456,'abc'）括号可有可无  >>> type(t)  <class 'tuple'>  >>> t[2]  'abc'  >>> tl=789,234,t #一个元组也可以作为另一个元组的一个元素  >>> tl[1]  234  >>> tl[2]  (123, 456, 'abc')  >>> tu=('a',345,(3.14,'b')) #元组作为另一个元组的元素时要加括号避免歧义  >>> tu[2]  (3.14, 'b')  >>> tu[1:]  (345, (3.14, 'b')) #元组属于序列的一种同样有切片操作  >>> tu\*2  ('a', 345, (3.14, 'b'), 'a', 345, (3.14, 'b'))  >>> tu  ('a', 345, (3.14, 'b')) |

元组定义后不能更改，也不能删除

与字符串一样，元组之间可以使用+号和\*进行运算

**4.列表（list）**

列表与元组类似，但是列表可以随时修改

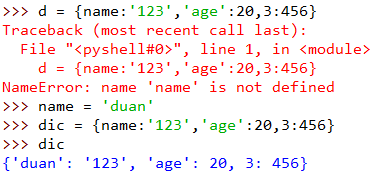
（1）列表相关方法

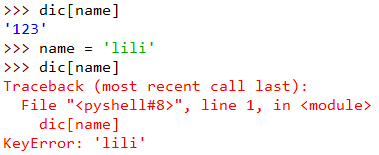
|  |  |
| --- | --- |
| 操作 | 含义 |
| <list>.append(x) | 将元素增加到列表的最后 |
| <list>.sort() | 将列表元素排序 |
| <list>.reverse() | 将列表元素反转 |
| <list>.index(x) | 返回第一次出现元素x的索引值 |
| <list>.insert(i,x) | 在位置i处插入新元素x |
| <list>.count(x) | 返回元素x在列表中的数量 |
| <list>.remove(x) | 删除列表中第一次出现的元素x |
| <list>.pop(i) | 取出列表中位置i的元素，并删除它 |

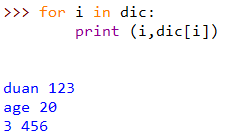
1. **字典 (dict)**

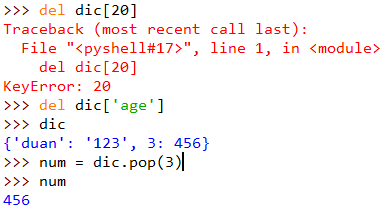
dict **= {**key1:value1,key2:value2.......}

key应为一个常量



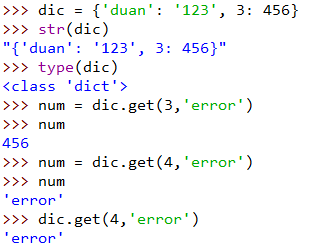






11

12



1. **函数**

基本形式

|  |
| --- |
| 1. def add(x, y): 2. """Add two numbers""" 3. a = x + y 4. return a |

因为python不用定义数据类型，所以函数最开始也不用写返回值类型，直接以def开头

与c++的不同之处：

1. 可以用关键词传参数

|  |
| --- |
| **def fun(**x**=**0, y**=**1, z**=**2**):** print**(**x, y, z**)** fun**(**y**=**3**)**  0 3 2  Process finished with exit code 0 |

2.传入元组

|  |
| --- |
| **def fun(**x**=**0, y**=**1, z**=**2**):** print**(**x, y, z**)** t **= (**4, 5, 6**)** fun**(\***t**)**  4 5 6  Process finished with exit code 0 |

注意星号不能少，否则报错，且只有一个星号

2.传入字典

|  |
| --- |
| **def fun(**x**=**0, y**=**1, z**=**2**):** print**(**x, y, z**)** d **= {'y':** 7, **'z':** 8**}** fun**(\*\***d**)**  0 7 8  Process finished with exit code 0 |

传字典需两个星号，且注意为‘y’，‘z’不能是y，z，因为之前说过key只能是常量，不能是变量

1. 返回多个值

|  |
| --- |
| **def fun(**x, y**):  return** y, x  num1 **=** 3 num2 **=** 5 num1, num2 **=** fun**(**num1, num2**)** #实现交换 print**(**num1, num2**)**  5 3  Process finished with exit code 0 |

1. 接受不定参数

|  |
| --- |
| **def fun(**x, **\***t**):** print**(**x**)** print**(**t**)** fun**(**1, 2**)** fun**(**3, 4, 5, 6**)**  1  (2,)  3  (4, 5, 6)  Process finished with exit code 0 |

多余参数相当于存放在元组t中

|  |
| --- |
| **def fun(**x, **\*\***d**):** print**(**x**)** print**(**d**)** fun**(**1, a**=**2, b**=**3**)**  1  {'a': 2, 'b': 3}  Process finished with exit code 0 |

以关键词形式传入参数并存放在字典中

|  |
| --- |
| **def fun(**x, **\***args, **\*\***kwargs**):** print**(**x, args, kwargs**)** fun**(**1, a**=**2, b**=**3, 4, 5**)**  File "F:/Python/fun", line 4  fun(1, a=2, b=3, 4, 5) ^  SyntaxError: positional argument follows keyword argument  Process finished with exit code 1 |

传入数据顺序应先按顺序赋值，多余的放在元组中，最后才能用关键字的形式传值，顺序不能乱。

|  |
| --- |
| **def fun(**x, **\***args, **\*\***kwargs**):** print**(**x, args, kwargs**)** fun**(**1, 4, 5, a**=**2, b**=**3, x**=**8**)**  Traceback (most recent call last):  File "F:/Python/fun", line 4, in <module>  fun(1, 4, 5, a=2, b=3, x=8)  TypeError: fun() got multiple values for argument 'x'  Process finished with exit code 1 |

注意以关键字传参时不要和顺序传参的参数名重复

1. lambda函数（匿名函数）

|  |
| --- |
| f **= lambda** x**:** x**\*\***3 # 相当于def f(x):  # return x\*\*3  print**(**f**(**2**))**  8  Process finished with exit code 0 |

三．应用小程序

（1）温度转换

|  |
| --- |
| 1. temperature = input("请输入温度及单位：") 2. if temperature[-1] == ('c' or 'C') : #注意括号一定要打，比较操作符比逻辑操作符优先级高 3. tran = float(temperature[0:-1]) \* 1.8 +32 4. print ("转换成华氏温度为：",tran, 'F') 5. elif temperature[-1] in （'f','F'） : #上面的和这种写法都可以 6. tran = ( float(temperature[0:-1]) - 32 ) / 1.8 7. print ("转换成摄氏温度为:%.2fC"%tran) #注意这里 8. else : 9. print ("输入有误") |

（2）猴子吃桃问题

|  |
| --- |
| 1. num = 1 2. for i in range (1,6) : 3. num = (num+1)\*2 4. print(num) 5. n = 1 6. for i in range (5,0,-1) : 7. n = (n+1)<<1 8. print(n) |

1. 附录
2. Python 3.x 保留字列表（33个）

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

1. 字符串处理方法