

1.在 python 中实践要求掌握的对象类型。

```
...  ← → week5
! environment.yml U use_of_str.py X use_of_bytes.py U
use_of_str.py > ...
11 b = id(p)
12 c = id(q)
13 print(b)
14 print(c)
15
16 p[0] = 2
17 print(b)
18 print(p)
19 print(q)
20
21 print(id(p))
22 print(id(q))
23
24 print(type(p))
25 print('instance(q, str):',instance(p,str))
26 print('instance(q, list):',instance(p,list))
27 print('instance(q, float):',instance(p,float))
28
29 try:
30     assert isinstance(p, str)
31 except AssertionError:
32     breakpoint()
33     print('type error')
34 print('over')
```

```
MINGW64:/e/D/jinrongbianch x + v
over
(week05)
朕真是帅极了@LAPTOP-CHRKOPNV MINGW64 /e/D/jinrongbianch
$ python use_of_str.py
nobody
2926216633424
2926216633424
2926214387968
2926214385984
2926214387968
[2, 2]
[1, 2]
2926214387968
2926214385984
<class 'list'>
instance(q, str): False
instance(q, list): True
instance(q, float): False
> e:\d\jinrongbiancheng\week5\use_of_str.py(33)<
-> print('type error')
(Pdb) l.
28
29 try:
30     assert isinstance(p, str)
31 except AssertionError:
32     breakpoint()
33     print('type error')
34     print('over')
[EOF]
(Pdb) p p
```

2.在 list 对象的实验中，尝试对 list 中的不同类型的对象进行条件处理，发现不符合条件的会被省略。

```
42 print(q)
43
44 p = [1, 2, 3]
45 q = [i**3 for i in p]
46 print(q)
47
48 p = ['m', 'n']
49 q = [i+'m' for i in p]
50 print(q)
51
52 p = [1, 'm', 3]
53 q = [i+2 for i in p if type(i) != str]
54 print(q)
```

```
3
3 1
a
unsupported operand type(s) for -: 'list' and 'list'
['n', 2]
['m', 2, 'm', 2]
[['m', 2], ['m', 2]]
[['n', 2], ['n', 2]]
[1, 8, 27]
['mm', 'nm']
[3, 5]
(week05)
朕真是帅极了@LAPTOP-CHRKOPNV MINGW64 /e/D/jinrongbiancheng/wee
$
```

3.在对字典 dict 进行实践的时候，发现 for 语句下输入类似的命令会有不同的结果：

```
use_of_dict.py X
use_of_dict.py > ...
1 a = {'a':2, 'b':3}
2 print(a)
3 print(type(a))
4
5 for i in a:
6     print(i)
7     print(a[i])
8 for b in a:
9     print(a[b])
```

```
3
(week05)
朕真是帅极了@LAPTOP-CHRKOPNV MINGW64 /e/D/jinrongbiancheng/wee
$ python use_of_dict.py
{'a': 2, 'b': 3}
<class 'dict'>
a
2
b
3
2
3
(week05)
朕真是帅极了@LAPTOP-CHRKOPNV MINGW64 /e/D/jinrongbiancheng/wee
```

问了豆包，知道这是因为两个 for 循环的代码逻辑不同：

在 Python 里，当使用 for 循环遍历字典时，默认是对字典的键进行遍历。所以在这个循

环中, i 会依次取到字典 a 的键 'a' 和 'b'。

当 i 为 'a' 时, print(i) 会输出键 'a', print(a[i]) 也就是 print(a['a']) 会输出对应的值 2。

当 i 为 'b' 时, print(i) 会输出键 'b', print(a[i]) 也就是 print(a['b']) 会输出对应的值 3。

所以, 第一个 for 循环的输出依次是键和对应的值, 整体输出为:

a

2

b

3

第二个循环同样, for 循环会遍历字典 a 的键, b 依次取到 'a' 和 'b'。不过在这个循环里, 只执行了 print(a[b]), 也就是只打印了键对应的值。

当 b 为 'a' 时, print(a[b]) 也就是 print(a['a']) 输出 2。

当 b 为 'b' 时, print(a[b]) 也就是 print(a['b']) 输出 3。

因此, 第二个 for 循环的输出是:

2

3

4.在对 set 进行实践的时候, 发现集合会自动去重, 即集合中不会有重复的元素。

```
use_of_tuple.py 1, U  use_of_set.py 1
use_of_set.py > ...
1 s = {1, 2, 3}
2 print(type(s))
3
4 try:
5     s = {1, [2], 'm'}
6 except TypeError as e:
7     print(e)
8
9 m = {1, 1, 3, 2, 1}
10 print(m)
11

$ python use_of_set.py
<class 'set'>
unhashable type: 'list'
{1, 2, 3}
(week05)
朕真是帅极了@LAPTOP-CHRKOPNV MINGW
$ python use_of_set.py
<class 'set'>
unhashable type: 'list'
{1, 2, 3}
(week05)
朕真是帅极了@LAPTOP-CHRKOPNV MINGW
$
```

5.