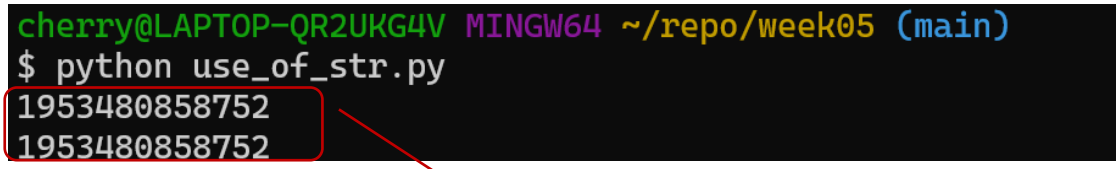
第 5 周 Python 对象类型

一、创建 use_of_str.py 文件

1. 在全局作用域 (global scope) 内尝试键入 (活学活用) Python 代码，亲手验证概念 (Proof of Concept, PoC)——不使用 def 定义函数
2. 对于任何对象，都可以传给以下内置函数 (built-in function) 用于检视 (inspect): Id()——返回对象在虚拟内存中的地址

每次运行的结果都不一样



x 和 y 的地址是一样的（都是字符串）

```python
a = [2, 5]
b = [2, 5]
x = id(a)
print(x)
y = id(b)
print(y)
```

此时 a 和 b 是列表

```
cherry@LAPTOP-QR2UKG4V MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
3234028919040
3234028917056
```

地址不一样，a 和 b 的值虽然是相等的，但二者不是同一个对象

```python
a = [2, 5]
b = [2, 5]
x = id(a)
print(x)
y = id(b)
print(y)
a[0] = 9
print(a)
print(b)
print(id(a))
print(id(b))
```

会是一样的运行结果吗？

```
$ python use_of_str.py
1177139812608
1177139810624
[9, 5]
[2, 5]
1177139812608
1177139810624
```

即便对对象的内容进行了修改，地址还是不变的

```python
a = [2, 5]
b = [2, 5]
c = "hello world"
x = id(a)
print(x)
y = id(b)
print(y)
a[0] = 9
print(a)
print(b)
print(id(a))
print(id(b))
print(type(a))
print(type(c))
```

```
$ python use_of_str.py
1363578525952
1363578523968
[9, 5]
[2, 5]
1363578525952
1363578523968
<class 'list'>
<class 'str'>
```

- Type()能够返回对象的类型

```
15    print(isinstance(a, str))
16    print(isinstance(c, str))
```

```
<class 'list'>
<class 'str'>
False
True
```

- isinstance 能够判断对象是否属于某个（某些）类型

```
print("dir of a:", dir(a))
```

```
dir of a: ['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__getstate__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
(week05)
```

Dir()生成的结果是一个列表，列表中是字符串，返回的是这个对象所支持的属性的名称列表

```
>>> print(32)
32
>>> print(str(32))
32
```

- str（）能够返回对象 print 是在终端显示的字符串——只有字符串能够 print 出来，32 是整数，转化为字符串再 print 出来

```
19    assert isinstance(a, str)
20    print("goodbye")
```

```
Traceback (most recent call last):
  File "C:\Users\cherry\repo\week05\use_of_str.py", line 19, in <module>
    assert isinstance(a, str)
           ^^^^^^^^^^^^^^^^^^
AssertionError
(week05)
```

- 可以利用 assert 查验某个表达式是否为真，由于 a 是列表，所以此处报错，并且报错后就会退出，后续的语句无法运行

```
18    try:
19        assert isinstance(a, str)
20    except AssertionError:
21        print("type error")
```

- 使用 try 语句也可以实现检查，同时能避免退出。Try 语句是一种流程控制语句，改变了本来会报错的流程，检查之后发现匹配 AssertionError，就继续运行后续的语句，不会报错退出。

```
20    except AssertionError:
21        breakpoint()
22        print("type error")
```

```
> c:\users\cherry\repo\week05\use_of_str.py(22)<module>()
-> print("type error")
(Pdb) l .
 17     print(isinstance(a, (str, list)))
 18     try:
 19         assert isinstance(a, str)
 20     except AssertionError:
 21         breakpoint()
 22  ->     print("type error")
 23     print("goodbye")
 24     print("dir of a:", dir(a))
[EOF]
(Pdb) p c
'hello world'
(Pdb) p isinstance(c, list)
False
```

- breakpoint()语句可以在代码的任何地方启动调试器

3. 对于要求掌握的对象类型（将来遇到新的对象类型也应该如此），我们首先应该熟悉如何通过 **表达式** (expression) 得到他们的 **实例** (instance)
   1) 字面值

```
1    print("字面值")
2    s = "university"
3    a = [33]
4    print(s)
5    print(a)
6    print(isinstance(s, str))
7    print(isinstance(a, str))
8    assert type(s) is str
```

```
$ python use_of_str.py
字面值
university
[33]
True
False
```

```
 9    print("f-string")
10    X = "5"
11    s = f"length:{X}"
12    print(s)
```

```
university
[33]
True
False
f-string
length:5
```

- f-string 语法的用法

```
14  s = "a\tb"
15  print("TAB", s)
16
17  s = "aaa\nbbbb"
18  print("next line", s)
```

a 和 b 之间会有三个空格

aaa 和 bbbb 之间会换行

```
26    print("初始化")
27    s = str()
28    print(s)
29    s = str([2, 3])
30    print(s)
```

```
初始化

[2, 3]
```

- str()语句可以使任意一个对象显示在终端时都以字符串的形式

```
assert str([2, 3]) == "[2, 3]"
```

以初始化形式得到的字符串

以字面值形式得到的字符串，二者是一样的

```
34    s = "2"
35    s = s * 5
36    print(s)
```

```
22222
```

- 运算值

```
41    s = "world"
42    print(s[3])
43    assert s[3] == "l"
44    print(s[1])
```

索引了 world 的第四个字母

索引值

字面值

```
53    s = "world"
54    u = s.upper()
55    print(s)
56    print(u)
```

返回了 s 这一字符串的大写实例，但是并没有改变 s（原始的字符串不可被修改）

```
62    s1 = "abc"
63    s2 = "xyz"
64    print(s2 + s1)
65    print(s2 - s1)
```

```
xyzabc
Traceback (most recent call last):
  File "C:\Users\cherry\repo\week05\use_of_str.py", line 65, in <module>
    print(s2 - s1)
          ~~~^~~~
TypeError: unsupported operand type(s) for -: 'str' and 'str'
(week05)
```

- 字符串可以进行加法运算但不能进行减法运算

- 字符串的相等：两个字符串中的内容一模一样

```
assert "abc" > "ABC"
print("123" > "a")
print("9" > ".")
```

```
False
True
```

- 字符是可以进行比较的，和代码位的排列顺序有关，在进行字符串的比较时，会逐个字母进行比较。

```
85  assert "abcde"
86  assert ""
```

```
False
Traceback (most recent call last):
  File "C:\Users\cherry\repo\week05\use_of_str.py", line 86, in <module>
    assert ""
           ^^
AssertionError
```

- 字符串中如果是空的就会报错

```
85  assert "yes"
86  assert not ""
87  s = "yes"
88  print(iter(s))
```

```
False
<str_ascii_iterator object at 0x000001812B050EB0>
```

可迭代：可以重复运行，循环

```
90  for c in s:
91      print(c)
```

如果 s 可迭代，都可以写成这种形式的 for 循环

```
(Pdb) g = iter(s)
(Pdb) p g
<str_ascii_iterator object at 0x000001E7A9C8C5B0>
(Pdb) p next(g)
'y'
(Pdb)
'e'
(Pdb)
's'
(Pdb)
*** StopIteration
```

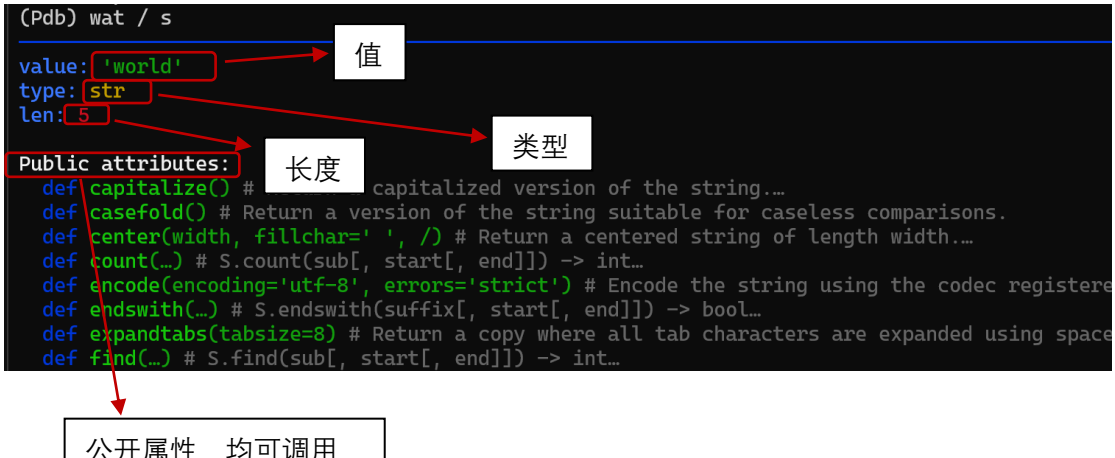
```
93    print(len(s))
94    try:
95        print(len(4))
96    except TypeError as e:
97        print(e)
```

```
3
object of type 'int' has no len()
```

- 字符串支持返回长度，数值不支持

```
 99    s = "world"
100    assert s[1:4] == "orl"
101    print(s[1:4])
```

- 字符串支持索引操作，【1：4】表示包含第一个，不包含第四个




```
(Pdb) p s
'world'
(Pdb) p s.translate({ord('rl'):ord('o')})
*** TypeError: ord() expected a character, but string of length 2 found
(Pdb) p s.translate({ord('r'):ord('o')})
'woold'
```

- s.translate 的用法

```
104    s = "hello world"
105    print(s.capitalize())
106    print(s)
```

```
Hello world
hello world
```

- 第一个字母大写

```
108    print("adh892".isalnum())
109    print("sjh*2".isalnum())
```

判断字符串是不是只由数字和字母组成

```
110  print("sd34".isidentifier())
111  print("34sd".isidentifier())
112  print("sd*34".isidentifier())
113  print("sd+34".isidentifier())
114  print("sd^34".isidentifier())
```

用来判断字符串可不可以作为变量名

```
116  q = ["a", "b", "c"]
117  print(";".join(q))
118  s = "2025-4-8"
119  print(s.split("-"))
```

分隔开

在列表中加入

2) 字节串

```
1  s = b"hello"
2  print(s)
3  print(s[0])
```

```
$ python use_of_bytes.py
b'hello'
104
```

Print 出的不是 h，而是 h 的序号。字节串意味着以二进制形式保存的一串字节

```
  6
  7     p = Path("D:\\Anaconda\\envs\\week05\\python.exe")
  8  -> breakpoint()
[EOF]
(Pdb) p p
WindowsPath('D:/Anaconda/envs/week05/python.exe')
(Pdb) p p.exists()
True
(Pdb) p p.is_file()
True
(Pdb) p p.is_dir()
False
```

判断这个地址的文件是否存在，是否为文件/文件夹

```
p = Path("D:\\Anaconda\\envs\\week05\\python.exe")
s = p.read_bytes()
print(len(s))
breakpoint()
```

```
$ python use_of_bytes.py
b'hello'
104
93184
```

- 查看字节长度

```
(Pdb) p s
b'print("\xe5\xad\x97\xe9\x9d\xa2\xe5\x80\xbc")\r\ns = "university"\r\
nce(s, str))\r\nprint(isinstance(a, str))\r\nassert type(s) is str\r\n
r\nprint(s)\r\n\r\ns = "a\\tb"\r\nprint("TAB", s)\r\n\r\ns = "aaa\\nbb
```

```
(Pdb) p s.decode()
'print("字面值")\r\ns = "university"\r\na = [33]\r\nprint(s)\r\nprint(a)\r\nprint(is
e(a, str))\r\nassert type(s) is str\r\nprint("f-string")\r\nX = "5"\r\ns = f"length:
r\nprint("TAB", s)\r\n\r\ns = "aaa\\nbbbb"\r\nprint("next line", s)\r\ns = """abc\r\
\r\nprint(s)\r\n\r\nprint("初始化")\r\ns = str()\r\nprint(s)\r\ns = str([2, 3])\r\np
```

```
15  s = b.decode()
16  assert isinstance(s, str)
17  b2 = s.encode()
18  assert isinstance(b2, bytes)
19  assert b2 == b
20  breakpoint()
```

- 用 decode 可以进行解码，将字节串解码为字符串，用 encode 进行编码，将字符串编码为字节串（二进制的）

```
21  s = "春天"
22  b = s.encode()
23  breakpoint()
```

```
(Pdb) p b
b'\xe6\x98\xa5\xe5\xa4\xa9'
(Pdb) p b[0]
230
```

对中文进行编码，十六进制

```
27  s = "wa哇😜"
28  print(s)
29  b = s.encode()
30  print(b)
31  breakpoint()
```

```
(Pdb) p b
b'wa\xe5\x93\x87\xf0\x9f\x98\x80'
(Pdb) p b[2:]
b'\xe5\x93\x87\xf0\x9f\x98\x80'
(Pdb) p b[2:].decode()
'哇😜'
(Pdb) p b[2:5].decode()
'哇'
```

- 对 emoji 进行编解码

3) 整数

```
z3 = c // b
print(z3)
assert z3 == 1
z4 = c % b
assert z4 == 3
```

除完的整数部分

除完的余数部分

```
(Pdb) for i in x:
*** IndentationError: expected an indented block after 'for' statement on line 1
(Pdb) for i in x: print(i)
*** TypeError: 'int' object is not iterable
(Pdb) p len(x)
*** TypeError: object of type 'int' has no len()
(Pdb) p x[0]
*** TypeError: 'int' object is not subscriptable
```

```
(Pdb) p x.to_bytes()
b'\t'
```

- 整数不可迭代循环、不可求长度、不可索取，但是可以转化为字节

4) 浮点数

```
4   b = "3.14"
5   print(type(b))
6   c = float("3.14")
7   print(type(c))
```

```
$ python use_of_float.py
<class 'float'>
<class 'str'>
<class 'float'>
```

- 浮点数的初始化

```
16  y = random.random()
17  print(y)
```

0 到 1 之间均匀分布的随机浮点数

- 尽量不要做浮点数是否相等的判断，因为 python 中的浮点数是四舍五入的

```
21  nan = float("nan")
22  print(nan + 3)
```

```
0.4262432264679239
nan
```

特殊的浮点数——缺失值，缺失值和任何数进行计算都等于缺失

```
27  pinf = float("inf")
28  print(1e200)
29  print(pinf > 1e200)
```

正无穷

科学计数法，表示 1×10^200

```
36  a = 1.1
37  b = 1.2
38  x = a + b
39  y = a - b
40  z1 = a * b
41  z2 = b / a
42  z2 = b // a
43  z3 = b % a
44  print(x, y, z1, z2, z3)
```

- 浮点数支持+、-、*、/、//、%运算符

5) Bool（布尔值）

- 类型是可以继承的，布尔值是整数的一个子类，只要整数有的性质布尔值都有

```
5  print(type(t))
6  print(isinstance(t, int))
```

```
<class 'bool'>
True
```

```
print(l1[-1])
print(l2[-2][1])
print(l2[-1][1])
```

索引了从右往左第二个元素中的从左往右的第二个元素

```
a = [1, 5]
b = ["a", "b"]
print(a + b)
print(b + a)
print(a + b == b + a)

a = [1, 2, 3]
b = [1, 2]
try:
    a - b
except TypeError as e:
    print(e)

print(b * 2)
```

- 列表支持+、*运算

```
a = [1, 2]
b = a * 3
a[0] = 7
print(a)
print(b)
```

给 a 的第一个元素赋值为 7

```
[1, 2, 1, 2]
[7, 2]
[1, 2, 1, 2, 1, 2]
```

改变了列表 a 中的元素，但没有改变列表 b

```
42    a = [1, 2]
43    b = [a] * 3
44    print(f"{b=}")
45    a[0] = 7
46    print(a)
47    print(b)
```

```
b=[[1, 2], [1, 2], [1, 2]]
[7, 2]
[[7, 2], [7, 2], [7, 2]]
```

修改了列表 a，对应的列表 b 也跟着被修改

```
49  a = [1, 2, 3]
50  b = [i**2 for i in a]
51  print(b)
52  c = [i + 2 for i in a]
53  print(c)
54  c = [i + 2 for i in a if i < 3]
55  print(c)
```

```
[1, 4, 9]
[3, 4, 5]
[3, 4]
```

- 列表的推导式

```
57    a = [1, 2]
58    b = [a] * 3
59    print(f"{b=}")
60    x = a.append(4)
61    print(x)
62    print(a)
63    print(b)
```

Append 表示在列表 a 后加上 4

```
b=[[1, 2], [1, 2], [1, 2]]
None
[1, 2, 4]
[[1, 2, 4], [1, 2, 4], [1, 2, 4]]
```

```
(Pdb) p a
[1, 2, 4]
(Pdb) p a.count(2)
1
```

计数某个值在列表中出现的次数

6) 字典（dict）

```
5  for a in d:
6      print(a)
```

```
<class 'dict'>
lili
cherry
jack
```

字典循环出来的是键

```
8  for a in d:
9      print(d[a])
```

```
lili
cherry
jack
163
168
188
```

- 循环的是键，再把值提取出来

```
14  l = [a for a in d.items()]
15  print(l)
```

推导式

```
[('lili', 163), ('cherry', 168), ('jack', 188)]
```

- 列表，每一个元组都是键-值对
- 列表和字典：列表使用整数来寻找元素，而字典是用键来找值，所以字典可以对键循环，也可以对值循环，还可以对键值对循环

```
(Pdb) p d
{'lili': 163, 'cherry': 168, 'jack': 188}
(Pdb) p d.get(cherry)
*** NameError: name 'cherry' is n
(Pdb) p d.get('cherry')
168
(Pdb) p d.get('a')
None
(Pdb) p d.get('a',0)
0
```

使用 get 来根据建找值

如果能找到值就给出值，如果找不到则给出默认值 0

```
(Pdb) p d
{'lili': 163, 'cherry': 168, 'jack': 188}
(Pdb) p d.pop('lili')
163
(Pdb) p d
{'cherry': 168, 'jack': 188}
```

- 弹出

```
(Pdb) p d
{'cherry': 168, 'jack': 188}
(Pdb) p d.setdefault('jack',0)
188
(Pdb) p d.setdefault('a',0)
0
(Pdb) p d
{'cherry': 168, 'jack': 188, 'a': 0}
```

Setdefault 不仅有 get 的功能，还会在最后加上这个没有的键值对

7) 元组

```
1  t = ("1", "a", 3.14)
2  print(type(t))
3  print(t)
4
```

```
$ python use_of_tuple.py
<class 'tuple'>
('1', 'a', 3.14)
```

- 圆括号，逗号分隔

```
5   print(len(t))
6   print(t[0])
7   print(t[1])
8   print(t[2])
```

```
('1', 'a', 3.14)
3
1
a
3.14
```

- 元组的索取、长度

元组和字典：元组是不可变的对象（不支持元素赋值），字典可对元素修改，但是字典的键必须是不可修改的对象

```
(Pdb) p t.count('1')
1
(Pdb) p t.index('a')
1
(Pdb) p t.index('aa')
*** ValueError: tuple.index(x): x not in tuple
```

- 元组的其他用法

```
18  q = [3, 1]
19
20  try:
21      d[q] = 2
22  except TypeError as e:
23      print(e)
24  print(d)
```

```
'tuple' object does not support item assignment
unhashable type: 'list'
{'aaa': 1, 3: 5}
```

- 列表不是不可变的对象，所以不能作为键进行赋值

```
26  t = (3, 1)
27  d[t] = 2
28  print(d)
```

```
{'aaa': 1, 3: 5}
{'aaa': 1, 3: 5, (3, 1): 2}
```

- 将列表更换为元组后，可作为字典的键进行赋值

```
31    t = 1, 2, 3.14
32    print(t)
33    print(type(t))
```

```
(1, 2, 3.14)
<class 'tuple'>
```

- 如果不会造成语法的歧义，在写元组时可省略括号

8) 集合

```
1    s = {1, 4, 7}
2    print(s)
3    print(type(s))
```

```
{1, 4, 7}
<class 'set'>
```

- 集合和字典很像，但是只有键没有值

```
14    q = [1, 2, 1, 2, 5, 1]
15    print(q)
16    s = set(q)
17    print(s)
```

```
[1, 2, 1, 2, 5, 1]
{1, 2, 5}
```

- 初始化（把重复的元素剔除）

```
24    s1 = {3, 2, 1}
25    print(s | s1)
26    print(s & s1)
27    print(s ^ s1)
```

```
{1, 2, 3, 5}
{1, 2}
{3, 5}
```

- 集合的运算（交、并、对称差）

```
5    print(p.exists())
6    print(p.absolute())
7    print(p.iterdir())
8    print(list(p.iterdir()))
```

9) Path

```
.
True
C:\Users\cherry\repo\week05
<generator object Path.iterdir at 0x000002DC0F4E35E0>
[WindowsPath('.git'), WindowsPath('.gitignore'), WindowsPath('environment.
ADME.md'), WindowsPath('use_of_bool.py'), WindowsPath('use_of_bytes.py'),
e_of_float.py'), WindowsPath('use_of_int.py'), WindowsPath('use_of_list.py
('use_of_set.py'), WindowsPath('use_of_str.py'), WindowsPath('use_of_tuple
```

- 显示绝对路径以及文件夹中所有的文件

```
10   p = Path("./data1")
11   print(p.exists())
12   p.mkdir()
13   print(p.exists())
14   print(p.is_dir())
```

- 判断该文件夹下是否有指定文件、创建文件

```
True
True
READEME.md
C:\Users\cherry\repo\week05\READEME.md
```

相对路径

绝对路径

```
Public attributes:
  anchor: str = 'C:\'
  drive: str = 'C:'
  name: str = 'READEME.md'
  parent: pathlib.WindowsPath = C:\Users\cherry\repo\week05
  parents: pathlib._PathParents = <WindowsPath.parents>
  parts: tuple = ('C:\\', 'Users', 'cherry', 'repo', 'week05', 'READEME.md')
  root: str = '\'
  stem: str = 'READEME'
  suffix: str = '.md'
  suffixes: list = […
```

- READEME.md 文件的属性

10) Datetime

```
1  from datetime import date, datetime, timedelta  # noqa: F401
2
3  print(date.today())
```

```
$ python use_of_datetime.py
2025-04-12
```

- 显示今天的日期

```
5    t1 = date.today()
6    t2 = date(2025, 11, 11)
7    print(t2 - t1)
```

初始化日期，年/月/日

```
$ python use_of_datetime.py
2025-04-12
213 days, 0:00:00
```

计算两个日期之间相差的天数

```
12   s1 = "2025-04-12"
13   s2 = "2025-10-18"
14   breakpoint()
```

```
(Pdb) p datetime.strptime(s1,'%Y-%m-%d')
datetime.datetime(2025, 4, 12, 0, 0)
(Pdb) p datetime.strptime(s1,'%Y-%m-%d').date()
datetime.date(2025, 4, 12)
```

- 将文本型的日期转化为可以进行计算的日期格式，但是文本型的日期必须满足“年-月-日“格式

```
(Pdb) p format(d1,'%a')
'Sat'
(Pdb) p format(d2,'%a')
'Sat'
(Pdb) p d1.strftime(%a)
*** SyntaxError: invalid syntax
(Pdb)  p d1.strftime('%a')
'Sat'
(Pdb) p d2.strftime('%a')
'Sat'
(Pdb) p format(d1,'%A')
'Saturday'
```

- 查看某一日期是周几

```
(Pdb) p d2.strftime('%Y年%m月%d日')
'2025年10月18日'
```

- 转化为想要的格式