

金融编程与计算

Week5——Python 对象类型（初级）

一、Fork 第 05 周打卡 仓库至你的名下，然后将你名下的这个仓库 Clone 到你的本地计算机

（一）fork

The screenshot shows the 'Create Fork' page on the Gitcode website. The URL is <https://gitcode.com/cueb-fintech/week05/fork/create>. The page title is '第05周打卡'. The 'Project Name' field is filled with '第05周打卡'. The 'Project Path' field is filled with 'TTATLib / week05'. The 'Project Description' field contains the text '学生 Fork 此仓库并通过 PR 提交第 5 周学习报告'. The 'Select Fork Branch' dropdown is set to 'main'. At the bottom, there are two buttons: '取消' (Cancel) and '创建Fork项目' (Create Fork Project).

（二）Clone

The screenshot shows the repository page for 'TTATLib/第05周打卡' on Gitcode. The URL is <https://gitcode.com/TTATLib/week05>. A 'Clone' modal is open, displaying instructions for cloning the repository. The modal includes the following text:

```
# 请确保本地完成了 Git 的全局配置
git config --global user.name TTATLib
git config --global user.email TTATLib@noreply.gitcode.com
```

Below this, there are tabs for 'HTTPS' and 'SSH'. The 'SSH' tab is selected. The modal also displays the SSH command to clone the repository:

```
git@gitcode.com:TTATLib/week05.git
```

```

(base) Administrator@PC-20220115W0JX MINGW64 ~/repo
$ pwd
/c/Users/Administrator/repo

(base) Administrator@PC-20220115W0JX MINGW64 ~/repo
$ git clone git@gitcode.com:TTATLib/week05.git
Cloning into 'week05'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 5 (delta 0), reused 5 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (5/5), 8.44 KiB | 2.11 MiB/s, done.

(base) Administrator@PC-20220115W0JX MINGW64 ~/repo
$ ls -l
total 24
drwxr-xr-x 1 Administrator 197121 0 3月 24 01:38 myproject/
drwxr-xr-x 1 Administrator 197121 0 3月 17 01:10 mywork/
drwxr-xr-x 1 Administrator 197121 0 3月 23 00:57 prj1/
-rw-r--r-- 1 Administrator 197121 0 3月 10 00:44 script1.py
drwxr-xr-x 1 Administrator 197121 0 3月 10 02:18 week01/
drwxr-xr-x 1 Administrator 197121 0 3月 17 01:17 week02/
drwxr-xr-x 1 Administrator 197121 0 3月 24 02:34 week03/
drwxr-xr-x 1 Administrator 197121 0 3月 30 21:54 week04/
drwxr-xr-x 1 Administrator 197121 0 4月 10 22:36 week05/
drwxr-xr-x 1 Administrator 197121 0 3月 19 21:03 work/

```

二、用 VS Code 打开项目目录，新建一个 `environment.yml` 文件，指定安装 `Python 3.12`，然后运行 `conda env create` 命令创建 Conda 环境

(一) 从 `myproject` 里 `cp` 过来 `environment.yml`

```

(base) Administrator@PC-20220115W0JX MINGW64 ~/repo
$ cd week05

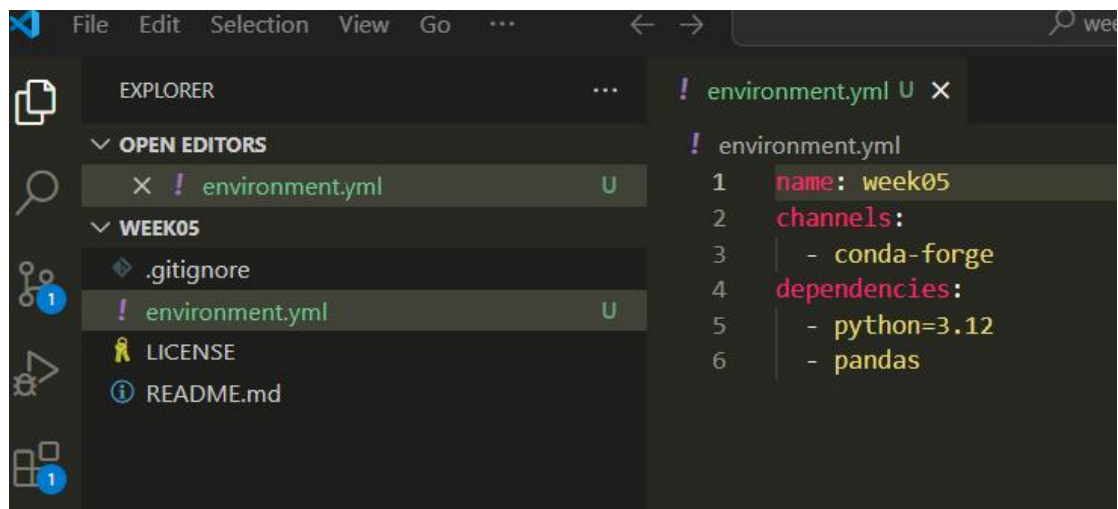
(base) Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ ls -l ../myproject
total 52114
-rw-r--r-- 1 Administrator 197121 87 3月 24 01:28 environment.yml
-rw-r--r-- 1 Administrator 197121 53362688 3月 24 01:53 EPA_SmartLocationDatabase_V3_3
-rw-r--r-- 1 Administrator 197121 713 3月 24 01:52 main.py

(base) Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ cat ../myproject/environment.yml
name: myproject
channels:
  - conda-forge
dependencies:
  - python=3.12
  - pandas
(base) Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ cp ../myproject/environment.yml
cp: missing destination file operand after '../myproject/environment.yml'
Try 'cp --help' for more information.

(base) Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ cp ../myproject/environment.yml ./

```

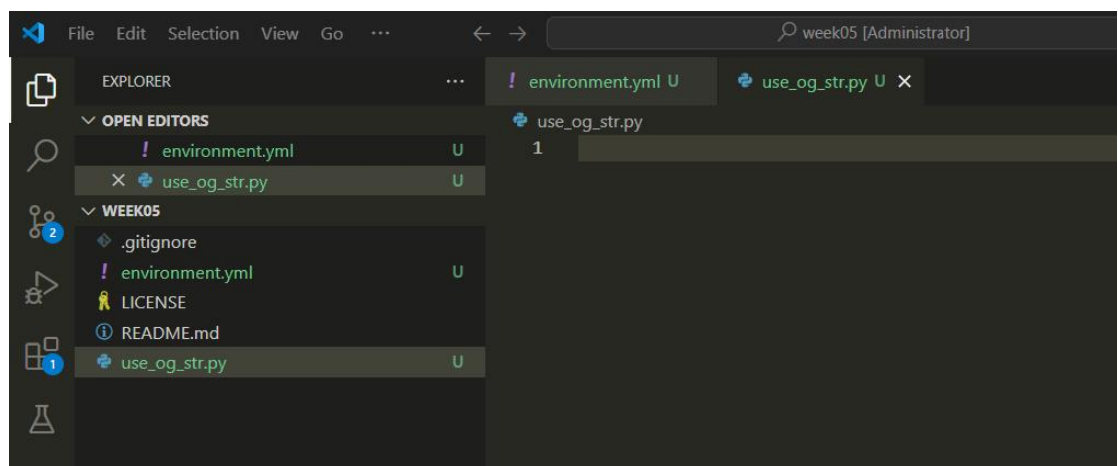
在 vscode 里把 name 改为 week05



四、逐个 创建 `use_of_{name}.py` 文件，其中 `{name}` 替换为上述要求掌握的对象类型，例如 `use_of_str.py`

注：字符串很重要！

（一）创建 `use_of_str.py`



调用 `print()` 函数将表达式 (expression) 输出到终端，查看结果是否符合预期

(二)id() -- 返回对象在虚拟内存中的地址（正整数），如果 `id(a) == id(b)`，那么 `a is b` (`is` 是个运算符)

```
! environment.yml U  use_of_str.py U x  use_of_bytes.py U
use_of_str.py > ...
1 a = 'hello'
2 print(a)
```

```
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
hello
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
```

```
! environment.yml U  use_of_str.py U x  use_of_bytes.py U
use_of_str.py > ...
1 a = 'hello'
2 x = id(a)
3 print(x)
```

```
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
2072577399936
(week05)
```

这个数字是 `a` 的 `id`

`id` 是用来显示地址的

```
! environment.yml U  use_of_str.py U x  use_of_bytes.py U
use_of_str.py > ...
1 a = 'hello'
2 b = 'hello'
3 x = id(a)
4 print(x)
5 y = id(b)
6 print(y)
```

Print 出来的两个 id 是一样的

```
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
1332698897536
1332698897536
(week05)
```

```
! environment.yml U use_of_str.py U
use_of_str.py > ...
1 a = [2, 5]
2 b = [2, 5]
3 x = id(a)
4 print(x)
5 y = id(b)
6 print(y)
```

这样 print 出来的 id 不一样

```
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
1949193804032
1949193802048
(week05)
```

```
use_of_str.py > ...  
1  a = [2, 5]  
2  b = [2, 5]  
3  x = id(a)  
4  print(x)  
5  y = id(b)  
6  print(y)  
7  a[0] = 9  
8  print(a)  
9  print(b)
```

```
(week05)  
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)  
$ python use_of_str.py  
2001610479872  
2001610477888  
[9, 5]  
[2, 5]
```



```

use_of_str.py > ...
1  a = [2, 5]
2  b = [2, 5]
3  x = id(a)
4  print(x)
5  y = id(b)
6  print(y)
7  a[0] = 9
8  print(a)
9  print(b)
10 print(id(a)) # is it same as x?
11 print(id(a)) # is it same as y?

```

```

(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
2510853445888
2510853443904
[9, 5]
[2, 5]
2510853445888
2510853445888

```

(二) **type()** -- 返回对象的类型

```

11 print(id(a)) # is it same as x?
12 print(type(a))

```

```

(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
3169762482432
3169762480448
[9, 5]
[2, 5]
3169762482432
3169762482432
<class 'list'>

```

(三) **isinstance()** -- 判断对象是否属于某个 (或某些) 类型

```
13 print('isinstance(a, str): ', isinstance(a, str))
```

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
2292972132608
2292972130624
[9, 5]
[2, 5]
2292972132608
2292972132608
<class 'list'>
isinstance(a, str): False
```

(四) `dir()` -- 返回对象所支持的属性 (attributes) 的名称列表

```
14 print('dir(a):', dir(a))
```

```
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
1519237339392
1519237337408
[9, 5]
[2, 5]
1519237339392
1519237339392
<class 'list'>
isinstance(a, str): False
dir(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']
```

(五) `str()` -- 返回对象 `print` 时要显示在终端的字符串

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



```

13 print('instance(a, str): ', instance(a, str))
14 print('instance(a, list): ', instance(a, list))

python use_of_str.py
2176185211136
2176185209152
[9, 5]
[2, 5]
2176185211136
2176185211136
<class 'list'>
instance(a, str): False
instance(a, list): True

```

a 是列表不是字符串，所以是 str 是 false，list 是 true

(六) assert 语句查验某个表达式 (expression) 为真, 否则报错

(AssertionError) 退出

```

assert instance(a, str)

```

如图，是 false，所以报错

```

$ python use_of_str.py
3242779089152
3242779087168
[9, 5]
[2, 5]
3242779089152
3242779089152
<class 'list'>
instance(a, str): False
instance(a, list): True
False
Traceback (most recent call last):
  File "C:\Users\Administrator\repo\week05\use_of_str.py", line 16, in <module>
    assert instance(a, str)
    ^^^^^^^^^^^^^^^^^^^^^
AssertionError

```

报错就会退出：

```

17 print('goodbye')

```

```

Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
2203587647744
2203587645760
[9, 5]
[2, 5]
2203587647744
2203587647744
<class 'list'>
isinstance(a, str): False
isinstance(a, list): True
False
Traceback (most recent call last):
  File "C:\Users\Administrator\repo\week05\use_of_str.py", line 16, in <module>
    assert isinstance(a, str)
    ^^^^^^^^^^^^^^^^^^^^^^^
AssertionError

```

如果没有报错:

```

16  assert isinstance(a, list)
17  print('goodbye')

```

```

(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
2672605731072
2672605729088
[9, 5]
[2, 5]
2672605731072
2672605731072
<class 'list'>
isinstance(a, str): False
isinstance(a, list): True
False
goodbye

```

三、对于 每一个 上述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), 一般包括以下途径:

字面值 (literal) (包括 f-string 语法)

推导式 (comprehension) (仅限 list、dict、set)

初始化 (init)

运算值 (operator)

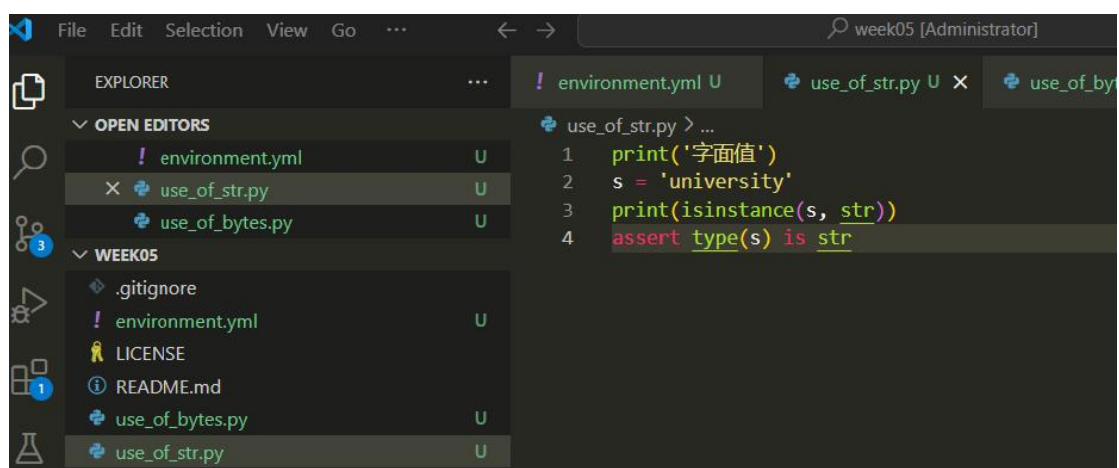
索引值 (subscription)

返回值 (return value of function/method call)

注: 字符串的实例, 是一个个具体的字符串

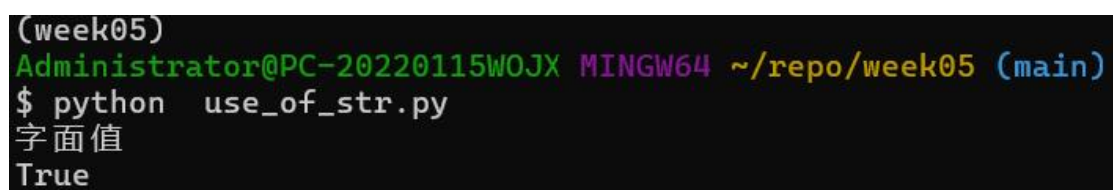
字符串的类型, 是他们共同的一个类型, 预先编好的关于类型的程序

(一) 字面值 (literal)



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'WEEK05' with files: .gitignore, environment.yml, LICENSE, README.md, use_of_bytes.py, and use_of_str.py. The code editor shows the content of 'use_of_str.py':

```
1 print('字面值')
2 s = 'university'
3 print(isinstance(s, str))
4 assert type(s) is str
```



The screenshot shows a terminal window with the following output:

```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
True
```

```
1  print('字面值')
2  s = 'university'
3  print(s)
4  print(isinstance(s, str))
5  assert type(s) is str
```

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
```

```
7  print('f-string')
8  x = 'Tom'
9  s = f'name: {x}'
10 print(s)
```

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
```

TAB 保证 a 到 b 之间有三个空格

```
12 s = 'a\tb'
13 print('TAB', s)
```

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
TAB a    b
```

\n: 换行的意思

```
15 s = 'aaa\nbbb'
16 print('New Line', s)
```

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
TAB a    b
New Line aaa
bbb
```

三个引号，换行符会原样保留，包括空格

```
20 s = """xyz
21 abc
22 |   eee
23 aaa
24 ""
25 print(s)
```

```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
TAB a   b
New Line aaa
bbb
xyz
abc
   eee
aaa
```

（二）推导式 (comprehension) (仅限 list、dict、set)

字符串里没有推导式，略过

（三）初始化

Str() 括号里可以没有东西

```
27 print('初始化')
28 s = str()
29 print(s)
```



```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
TAB a    b
New Line aaa
bbb
xyz
abc
    eee
aaa

初始化
```

```
29     print(s)
30     s = str([5, 8, 2])
```

```

(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
TAB a    b
New Line aaa
bbb
xyz
abc
    eee
aaa

初始化

[5, 8, 2]

```

没有报错，说明这两个相等

注：assert 的含义

在 Python 中，`assert` 是一个用于调试和程序验证的关键字，称为 **断言**。它的作用是检查某个条件是否为 `True`，若条件为 `False`，则抛出 `AssertionError` 异常，中断程序执行，帮助开发者快速定位逻辑错误。

```

32
33  assert str([5, 8, 2]) == '[5, 8, 2]'

```

```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
TAB a    b
New Line aaa
bbb
xyz
abc
    eee
aaa

初始化

[5, 8, 2]
```

```
34     assert str(1.1 + 2.2) == '3.3'
```

```
初始化

[5, 8, 2]
Traceback (most recent call last):
  File "C:\Users\Administrator\repo\week05\use_of_str.py", line 34, in <module>
    assert str(1.1 + 2.2) == '3.3'
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError
```

发现报错了，为什么报错？

用 `python -m pdb use_of_str.py` 检查

```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python -m pdb use_of_str.py
> c:\users\administrator\repo\week05\use_of_str.py(1)<module>()
-> print('字面值')
(Pdb) c
```

输入 `c` 继续（continue）直到报错为止

如图，第 34 行报错了

```
AssertionError
Uncaught exception. Entering post mortem debugging
Running 'cont' or 'step' will restart the program
> c:\users\administrator\repo\week05\use_of_str.py(34)<module>()
-> assert str(1.1 + 2.2) == '3.3'
(Pdb) l .
29     print(s)
30     s = str([5, 8, 2])
31     print(s)
32
33     assert str([5, 8, 2]) == '[5, 8, 2]'
34 -> assert str(1.1 + 2.2) == '3.3'
[EOF]
(Pdb)
```

```
(Pdb) p str(1.1 + 2.2)
'3.3000000000000003'
(Pdb)
```

因为计算机里面是二进制保存

输入 q 退掉 python

```
(Pdb) p str(1.1 + 2.2)
'3.3000000000000003'
(Pdb) q
Post mortem debugger finished. The C:\Users\Administrator\repo\week05\use_of_str.
> c:\users\administrator\repo\week05\use_of_str.py(1)<module>()
-> print('字面值')
(Pdb) q
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
TAB a b
New Line aaa
bbb
xyz
abc
eee
aaa

初始化
[5, 8, 2]
```

(四) 运算值 (operator)

```
38     s = '='
39     s = s * 20
40     print(s)
```

出来 20 个等号:

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (m
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
TAB a    b
New Line aaa
bbb
xyz
abc
    eee
aaa

初始化

[5, 8, 2]
=====
```

加个断点 breakpoint

```
36     assert str() == ''
37
38     breakpoint()
39     s = '='
40     s = s * 20
41     print(s)
```

```
38 breakpoint()
39 s = '='
40 x = id(s)
41 s = s * 20
42 y = id(s)
43 print(s)
44 assert x != y
```

把断点去掉后再运行

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
TAB a    b
New Line aaa
bbb
xyz
abc
    eee
aaa

初始化

[5, 8, 2]
=====
```

(五) 索引值 (subscription)


```
45     s = 'hello'
46     assert s[3] == 'l'
```

如上图：l 字符串，即'l'是字面值，s[3]是索引值，对 s 这个对象进行索引操作

运行不报错，（hello 的第三个字符就是 l）

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
字面值
university
True
f-string
name: Tom
TAB a    b
New Line aaa
bbb
xyz
abc
    eee
aaa

初始化

[5, 8, 2]
=====
```

-1 表示最后一个，即 o

```
46     assert s[3] == 'l'
47     assert s[-1] == 'o'
```

:3 表示从左边拉三个字符串，即 hel

```
48     assert s[:3] == 'hel'
```

[5]就会报错，因为没有五号

```
50     print(s[5])
```

```
[5, 8, 2]
=====
Traceback (most recent call last):
  File "C:\Users\Administrator\repo\week05\use_of_str.py", line 50, in <
    print(s[5])
    ~^^^
IndexError: string index out of range
```

```
50     try:
51         s[5]
52     except IndentationError as e:
53         print(e)
```

```
54     s = 'hello'
55     u = s.upper()
56     print(u)
```

出现大写的 HELLO:

初始化

```
[5, 8, 2]
=====
HELLO
(5, 8, 2)
```

```

54 s = 'hello'
55 u = s.upper()
56 print(u)
57 print(s)

```

```

[5, 8, 2]
=====
HELLO
hello

```

```

59 t = 'name: {}, age {}'
60 print(t)

```

```

[5, 8, 2]
=====
HELLO
hello
name: {}, age {}

```

```

61 t1 = t.format('Jack', 21)
62 print(t1)

```

```

HELLO
hello
name: {}, age {}
name: Jack, age 21

```

五、对于 每一个 上述要求掌握的对象类型 (将来遇到新的对象类型也应该如此), 我们也要尝试验证其以下几个方面的 属性

(attributes):

- 对数学运算符 (+、-、*、**、/、//、%、@) 有没有支持
- 如何判断相等 (==) ◦ 对于比较运算符 (>、<、>=、<=) 有没有支持
- 什么值被当作 True，什么值被当作 False
- 是否可迭代 (iterable)，如何做迭代 (for 循环)
- 是否支持返回长度 (len)
- 是否 (如何) 支持索引操作 (subscription) ([] 运算符)
- 拥有哪些常用方法 (method) 可供调用 (() 运算符)

(一) 对数学运算符 (+、-、*、**、/、//、%、@) 有没有支持

```

64 s1 = 'abc'
65 s2 = 'ghi'
66 s = s1 + s2
67 assert s == 'abcghi'
68 print(s2 + s1)

```

显示 ghiabc

```

[5, 8, 2]
=====
HELLO
hello
name: {}, age {}
name: Jack, age 21
ghiabc
(week05)

```

```

72     s = ' *= '
73     s = s * 10
74     print(s)

```

```

name: {}, age {}
name: Jack, age 21
ghiabc
=====
(week05)

```

(二) 如何判断相等 (==) 。对于比较运算符 (>、<、>=、<=) 有没有支持

两个字符串的内容一模一样就是相等

```

75     assert s == 'aaaa'

```

```

=====
Traceback (most recent call last):
  File "C:\Users\Administrator\repo\week05\use_of_str.py", line 75, in <module>
    assert s == 'aaaa'
    ^^^^^^^^^^^^^^^
AssertionError
(week05)
Administrator@PC-20220115WQJY MINGW64 ~/repo/week05 (main)

```

字符串比较大小

```

76
77     print('abc' > 'ABC')

```

[illegible]

字符串里每一个字符都有代码位，位置靠前就是小，靠后就大

“美国标准信息交换码”

```
76 print('abc' > 'ABC')
77 print('123' > 'abcd')
78 print('9' > '.')
79 print(': ' > '9')
80
```

True
False
True
True

（三）什么值被当作 **True**，什么值被当作 **False**

| | |
|---------------|-------|
| | True |
| | False |
| | True |
| | True |
| assert 'book' | True |
| | False |

```
assert '' AssertionError
```

只要字符串长度不为 0 就是 true，为 0 就报错

(四) 是否可迭代 (iterable), 如何做迭代 (for 循环)

用 iter 不报错就是可迭代


```

85
86 s = 'book'
87 print(iter(s))

```

```

True
False
True
True
True
True
False
<str_ascii_iterator object at 0x000001D86D75CF10>

```

Iterator 迭代器，可迭代

如何做迭代？

```

88
89 for c in s:
90     print(c)

```

b
o
o
k

(五) 是否支持返回长度 (len)

```

print(len(s))

```

b
o
o
k
4

(六) 是否 (如何) 支持索引操作 (subscription) ([] 运算符)

```

s = 'book'
assert s[1:3] == 'oo'

```

注: [1:3] 3-1=2, 两个字符

(七) 拥有哪些常用方法 (method) 可供调用 (()) 运算符)

```
s = 'the book of why'
print(s.capitalize())
print(s)
breakpoint()
```

4
The book of why
the book of why
--Return--

Count

```
s = 'the book of why took nooo'
print(s.capitalize())
print(s)
print(s.count('oo'))
breakpoint()
```

```
(Pdb) p s
'the book of why took nooo'
(Pdb) p s.endswith('why')
False
(Pdb) p s.endswith('nooo')
True
```

```
print('abc123'.isalnum())
breakpoint()
```

3
True
--Return--

Isidentifier: 判断字符串能否作为标识符, 能就是 true

```
print('abc123'.isidentifier())
```

```
q = ['rose', 'jack', 'bob']
print('.'.join(q))
s = 'rose:jack:bob'
print(s.split(":"))
assert s.partition(':') == ("rose", ":", "jack:bob")
```

```
rose:jack:bob
['rose', 'jack', 'bob']
(week05)
```

字节串 bytes

```
use_of_bytes.py > ...
1 s = b'hello'
2 print(s)
```

```
(week05)
Administrator@PC-20220115W0JX
$ python use_of_bytes.py
b'hello'
```

```
use_of_bytes.py > ...
1  from pathlib import Path
2
3  s = b'hello'
4  print(s)
5
6  p = Path("/d/anaconda/envs/week05/python")
7  breakpoint()
```

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_bytes.py
b'hello'
--Return--
> c:\users\administrator\repo\week05\use_of_bytes.py(7)<module>()->None
-> breakpoint()
(Pdb) p p
WindowsPath('/d/anaconda/envs/week05/python')
```

把路径换成\\

```
6 p = Path("d:\\anaconda\\envs\\week05\\python.exe")
7 breakpoint()
```

```
> c:\users\administrator\repo\week05\use_of_bytes.py(7)<module>()->None
-> breakpoint()
(Pdb) p p
WindowsPath('d:/anaconda/envs/week05/python.exe')
(Pdb) p p.exists()
True
```

把字节全都读出来: `s = p.read_bytes()`

[illegible]

```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ ls -l /d/anaconda/envs/week05/python
-rwxr-xr-x 5 Administrator 197121 93184 3月 5 06:39 /d/anaconda/envs/week05/python*
```

```
10 p = Path("environment.yaml")
11 s = p.read_bytes()
12 print(s)
13 breakpoint()
```

```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_bytes.py
b'hello'
> c:\users\administrator\repo\week05\use_of_bytes.py(10)<module>()
-> p = Path("environment.yaml")
(Pdb)
```

关系：

编码就是本来是个字符串的东西，经过编码后变成二进制的东西，这个二进制的东西就是字节串。

字符串可以编码成字节串，字节串解码可以变成字符串

```
(Pdb) p 2**4
16
(Pdb) p b'\x5a'
b'Z'
```

整数 (int)

一、运算

```
use_of_int.py > ...
1  i = 42
2  x = 8
3  y = 5
4  z = x + y
5
6
7  x = 3
8  y = 8
9  assert y // x == 2
```

//整除的意思

```
x = 5
y = 17
assert y // x == 3
assert y % x == 2
```

%余除的意思

二、true or false

```
11
12  assert 5
13
14  try:
15      |   assert 0
16  except AssertionError as e:
17      |   print(e)
18
```

Try: 防报错


```
print(type(e))
```

```
(week05)
Administrator@PC-20220115W0JX MINGW64
$ python use_of_int.py
<class 'AssertionError'>
(week05)
```

三、迭代 (iter)

加个断点 breakpoint ()

```
(Pdb) l
14     try:
15         assert 0
16     except AssertionError as e:
17         print(type(e))
18
19 -> breakpoint()
[EOF]
(Pdb) p x
5
(Pdb) for i in x
*** SyntaxError: expected ':'
(Pdb) for i in x:print(i)
*** TypeError: 'int' object is not iterable
(Pdb) p iter(x)
*** TypeError: 'int' object is not iterable
(Pdb)
```

结论：整数不可循环迭代

四、是否支持返回长度 (len)

```
(Pdb) p len(x)
*** TypeError: object of type 'int' has no len()
(Pdb)
```

不支持

整数转化成字节：

```
(Pdb) p x
5
(Pdb) p x.to_bytes()
b'\x05'
```

```
(Pdb) p x
65535
(Pdb) p x.to_bytes()
*** OverflowError: int too big to convert
(Pdb)
```

如上图：显示整数太大了，如下图：两个字节就可以

```
(Pdb) p x.to_bytes(2)
b'\xff\xff'
```

浮点数（float）

```
use_of_float.py > ...
1 x = 3.14
2 print(type(x))
3
```

运行出来的 class（类型）为 float（浮点数）：

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (ma
$ python use_of_float.py
<class 'float'>
```

```
x = 3.14
print(type(x))

y = float("3.14")
print(type(y))

assert x == y
```

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/re
$ python use_of_float.py
<class 'float'>
<class 'float'>
```

没有报错，两者相等成立

```
x = 5/3
print(x, type(x))
```

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_float.py
<class 'float'>
<class 'float'>
1.6666666666666667 <class 'float'>
```

浮点数，用有限个比特保存

```
use_of_float.py > ...
1  import random
2
3  x = 3.14
4  print(type(x))
5
6  y = float("3.14")
7  print(type(y))
8
9  assert x == y
10
11 x = 5/3
12 print(x, type(x))
13
14 x = random.random()
15 print(x)
```

Random: 在 0-1 中均匀抽取的随机数

```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_float.py
<class 'float'>
<class 'float'>
1.6666666666666667 <class 'float'>
0.131750305767395
```

缺失值跟任何数字运算得出的结果都是缺失值（nan）

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

```
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_float.py
<class 'float'>
<class 'float'>
1.6666666666666667 <class 'float'>
0.18858035990428157
nan
```

```
nan = float("nan")
print(nan + 3)
print(nan > 3)
print(nan < 3)
print(nan == 3)
```

```
nan
False
False
False
```

```
pinf = float("inf")
print(3.14e2)
```

```
nan
False
False
False
314.0
```

```
pinf = float("inf")
print(3.14e2)
print(3.14e-2)
```

```
False
314.0
0.0314
```

注：pinf 是正无穷的意思

正无穷大于任何浮点数

```
print(pinf > 1e200) True
```

正无穷等于正无穷

```
print(pinf == pinf) True
```

正无穷 pinf

负无穷 ninf

```
ninf = float('-inf')
print(ninf)
-inf
```

布尔值 (bool)

```
use_of_bool.py > ...
1 t = True
2 f = False
3 print(t, f)
```

```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (n
$ python use_of_bool.py
True False
```

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

```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05
$ python use_of_bool.py
True False
<class 'bool'>
True
(week05)
```

布尔值继承了整数，整数有的性质布尔值都有

布尔值是一种特殊的整数

四大容器类型

列表（list）字典（dict）元组（tuple）集合（set）

一、列表（list）

```
use_of_list.py > ...
1  l = [1, 5, 'abc']
2  print(l)
3
```

```
(week05)
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ python use_of_list.py
[1, 5, 'abc']
```

```

print(l[0])
print(l[1])
print(l[2])

print(l[3])

```

报错：索引错误，然后 try

```

try:
    print(l[3])
except IndexError as e:
    print(e)

```

```

(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)
$ python use_of_list.py
[1, 5, 'abc']
1
5
abc
list index out of range

```

```

print(l[-1])
print(l[-1][-1])

```

list index out of range
abc
c

列表可以比较大小

```

a = [2, 5]
b = ['a', 'c']
print(a + b)
print(b + a)
print(a + b == b + a)

```

[2, 5, 'a', 'c']
['a', 'c', 2, 5]
False


```
a = [2, 5]
b = [5]
print(a - b)
```

报错：不支持减号

```
a = [2, 5]
print(a * 3)
```

```
unsupported operand type(s) for -: 'list' and 'list'
[2, 5, 2, 5, 2, 5]
```

支持乘法

```
a = [2, 5]
b = a * 3
a[0] = 9
print(a)
```

[9, 5]

```
print(f"{b=}")
a[0] = 9
print(a)
print(b)
```

```
a = [2, 5]
b = [a] * 3
```

```
unsupported operand type(s)
[2, 5, 2, 5, 2, 5]
b=[2, 5, 2, 5, 2, 5]
[9, 5]
[2, 5, 2, 5, 2, 5]
```

列表重复三次：

```
b = [a] * 3
print(f"{b=}")
```

```
b=[2, 5, 2, 5, 2, 5]
[9, 5]
[2, 5, 2, 5, 2, 5]
b=[[2, 5], [2, 5], [2, 5]]
```

```

a = [2, 5]
b = [a] * 3
print(f"{b=}")
a[0] = 9
print(a)
print(b)

```

```

[2, 5, 2, 5, 2, 5]
b=[2, 5, 2, 5, 2, 5]
[9, 5]
[2, 5, 2, 5, 2, 5]
b=[[2, 5], [2, 5], [2, 5]]
[9, 5]
[[9, 5], [9, 5], [9, 5]]

```

```

a = [2, 5, 3]
b = [i**2 for i in a]
print(b)

```

```

[[9, 5], [9, 5], [9, 5]]
[4, 25, 9]

```

加条件 if

```

b = [i**2 for i in a if i < 4]

```

```

[4, 25, 9]
[4, 9]

```

只有 4, 9

```

a = [2, 5]
b = [a] * 3
print(f"{b=}")
x = a.append(4)
print(x)
print(a)
print(b)

```

```

None
[2, 5, 4]
[[2, 5, 4], [2, 5, 4], [2, 5, 4]]

```

二、字典 (dict) 最重要

```
use_of_dict.py > ...  
1 d = {'a': 1, 'bb':5, 'cat': 3}  
2 print(d)
```

```
Administrator@PC-20220115W0JX MINGW64 ~/re  
$ python use_of_dict.py  
{'a': 1, 'bb': 5, 'cat': 3}
```

```
d = {'a': 1, 'bb':5, 'cat': 3}  
print(d)  
print(type(d))
```

```
Administrator@PC-20220115W0JX MINGW64 ~/re  
$ python use_of_dict.py  
{'a': 1, 'bb': 5, 'cat': 3}  
<class 'dict'>
```

```
(Pdb) p hash('a')  
-9190097581080323667  
(Pdb) p hash('a')  
-9190097581080323667  
(Pdb) p hash('a')  
-9190097581080323667  
(Pdb) p hash('bb')  
-2630201785389550365  
(Pdb) p hash('cat')  
3416257835035409226  
(Pdb) p hash(1)  
1  
(Pdb) p hash(2)  
2
```

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

a
bb
cat

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

1
5
3

```
for a in d.values():
    print(a)

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

键值构成的一个元组

```
[('a', 1), ('bb', 5), ('cat', 3)]
```

```
for k, v in d.items():
    print(k, v)
```

a 1
bb 5
cat 3

报错输入 get

```
(Pdb) p d
{'a': 1, 'bb': 5, 'cat': 3}
(Pdb) p d['bbb']
*** KeyError: 'bbb'
(Pdb) p d.get('bb')
5
(Pdb) p d.get('bbb')
None
```

元组 (tuple)

```
use_of_tuple.py > ...  
1  t = (1, 'a', 3.14)  
2  print(t)  
3  print(type(t))  
4
```

```
(week05)  
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)  
$ python use_of_tuple.py  
(1, 'a', 3.14)  
<class 'tuple'>
```

元组和列表很像

区别：元组比列表更加简单，元组是一个不可变的对象

```
(week05)  
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)  
$ python use_of_tuple.py  
(1, 'a', 3.14)  
<class 'tuple'>  
1  
a  
3.14
```

```
print(t[0])  
print(t[1])  
print(t[2])
```

元组不支持元素的赋值，如 `t = [9]` 运行会报错

列表不会报错：

```
use_of_tuple.py > ...  
1  t = [1, 'a', 3.14]  
2  print(t)  
3  print(type(t))  
4  
5  print(t[0])  
6  print(t[1])  
7  print(t[2])  
8  
9  t[0] = 9
```

```
Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)  
$ python use_of_tuple.py  
[1, 'a', 3.14]  
<class 'list'>  
1  
a  
3.14
```

```
d = {}  
d['abc'] = 5  
print(d)
```

```
{'abc': 5}
```

```
d = {}  
d['abc'] = 5  
d[7] = 100  
print(d)
```

```
{'abc': 5, 7: 100}
```

列表不可算哈希值

```
d = {}  
d['abc'] = 5  
d[7] = 100  
q = [3, 1]  
d[q] = 21  
print(d)
```

```
d[q] = 21  
~^^^
```

```
TypeError: unhashable type: 'list'
```

```
t = (3, 1)  
d[t] = 21  
print(d)
```

```
unhashable type: 'list'  
{'abc': 5, 7: 100, (3, 1): 21}
```

```
t = 1, 4, 0, 2  
print(t)  
print(type(t))
```

```
21  
(1, 4, 0, 2)  
<class 'tuple'>
```

元组不可变，列表、字典可变，其他的一堆都不可变

集合 (set)

```
use_of_set.py > ...  
1 s = {1, 4, 7}  
2 print(s)  
3 print(type(s))
```

```
(week05)  
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)  
$ python use_of_set.py  
{1, 4, 7}  
<class 'set'>
```

集合可以达到将重复的东西剔除掉的效果:

```
q = [1, 2, 1, 2, 5, 1]  
print(q)  
s = set(q)  
print(s)
```

```
<class 'set'>  
[1, 2, 1, 2, 5, 1]  
{1, 2, 5}
```

```
s = {5, 2, 1, 2, 2, 1}  
print(s)  
print(2 in s)  
print(3 in s)
```

```
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05 (main)  
$ python use_of_set.py  
{1, 4, 7}  
<class 'set'>  
[1, 2, 1, 2, 5, 1]  
{1, 2, 5}  
{1, 2, 5}  
True  
False  
(week05)
```


并集

```
s2 = {3, 2, 3}
print(s | s2)
```

{1, 2, 3, 5}

交集

```
print(s & s2)
```

{2}

```
use_of_path.py > ...
1  from pathlib import Path
2
3  p = Path('.')
4  print(p)
5
```

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week
$ python use_of_path.py
.
```

```
p = Path('.')
print(p)
print(p.exists())
```

Administrator@PC-20220115WOJX MINGW64 ~/repo/week
\$ python use_of_path.py
.
True

```
use_of_path.py > ...
1  from pathlib import Path
2  from pprint import pprint
3
4  p = Path('.')
5  print(p)
6  print(p.exists())
7  print(p.absolute())
8  pprint(list(p.iterdir()))
```

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/week05
$ python use_of_path.py
.
True
C:\Users\Administrator\repo\week05
[WindowsPath('.git'),
 WindowsPath('.gitignore'),
 WindowsPath('environment.yml'),
 WindowsPath('LICENSE'),
 WindowsPath('README.md'),
 WindowsPath('use_of_bool.py'),
 WindowsPath('use_of_bytes.py'),
 WindowsPath('use_of_dict.py'),
 WindowsPath('use_of_float.py'),
 WindowsPath('use_of_int.py'),
 WindowsPath('use_of_list.py'),
 WindowsPath('use_of_path.py'),
 WindowsPath('use_of_set.py'),
 WindowsPath('use_of_str.py'),
 WindowsPath('use_of_tuple.py')]
```

```
use_of_path.py > ...
1  from pathlib import Path
2  from pprint import pprint
3
4  p = Path('.')
5  print(p)
6  print(p.exists())
7  print(p.absolute())
8  pprint(list(p.iterdir()))
9
10 p = Path('./data1')
11 print(p.exists())
12 p.mkdir()
13 print(p.exists())
14 print(p.is_dir())
```

False
True
True

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

```
(week05)
Administrator@PC-20220115WOJX MINGW64 ~/repo/
$ python use_of_datetime.py
2025-04-13
```

计算今天到双十一相差多少天

```
use_of_datetime.py > ...
1  from datetime import date, datetime
2
3  t1 = date.today()
4  t2 = date(2025, 11, 11)
5  print(t2 - t1)
```

211 days,

```
t1 = date.today()
t2 = date(2025, 11, 11)
td = t2 - t1
print(td)
print(type(td))
```

```
211 days, 0:00:00
<class 'datetime.timedelta'>
(week05)
```

```
s1 = "2024-08-05"
s2 = "2024-10-05"
d1 = datetime.strptime(s1, "%Y-%m-%d")
d2 = datetime.strptime(s2, "%Y-%m-%d")
print(d1)
print(d2)
```

```
2024-08-05 00:00:00
2024-10-05 00:00:00
```