

```

$ git clone git@github.com:dawnchun/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 | 1.69 MiB/s, done.

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo
$ ls -l
total 21
drwxr-xr-x 1 LENOVO 197121  0 Mar 20 22:48 myproject/
drwxr-xr-x 1 LENOVO 197121  0 Mar 16 22:52 mywork/
drwxr-xr-x 1 LENOVO 197121  0 Mar 19 23:41 prj1/
-rw-r--r-- 1 LENOVO 197121 316 Mar 11 15:47 script1.py
drwxr-xr-x 1 LENOVO 197121  0 Mar 16 23:22 week01/
drwxr-xr-x 1 LENOVO 197121  0 Mar 16 23:22 week02/
drwxr-xr-x 1 LENOVO 197121  0 Mar 24 15:10 week03/
drwxr-xr-x 1 LENOVO 197121  0 Mar 29 00:22 week04/
drwxr-xr-x 1 LENOVO 197121  0 Apr  9 16:06 week05/

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo
$ cd week05

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ pwd
/c/Users/LENOVO/repo/week05

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ cat week04/eviroment.yml
cat: week04/eviroment.yml: No such file or directory

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ cat week04/environment.yml
cat: week04/environment.yml: No such file or directory

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ cd ..

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo
$ cat week04/environment.yml
name: week04
channels:
  - conda-forge
dependencies:
  - python=3.12
  - wat-inspector

```

```

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ cd ..

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo
$ cat week04/environment.yml
name: week04
channels:
  - conda-forge
dependencies:
  - python=3.12
  - wat-inspector
(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo
$ cp week04/environment.yml week05/

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo
$ ls -l week05
total 25
-rw-r--r-- 1 LENOVO 197121 18805 Apr  9 16:06 LICENSE
-rw-r--r-- 1 LENOVO 197121 2239 Apr  9 16:06 README.md
-rw-r--r-- 1 LENOVO 197121   91 Apr  9 16:18 environment.yml

```

```

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo
$ cd week05

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ conda env create
D:\anaconda\Lib\argparse.py:2006: FutureWarning: `remote_definition` is deprec
ed and will be removed in 25.9. Use `conda env create --file=URL` instead.
  action(self, namespace, argument_values, option_string)
Retrieving notices: ...working... done
Channels:
  - conda-forge
  - defaults
  - https://repo.anaconda.com/pkgs/main
  - https://repo.anaconda.com/pkgs/r
  - https://repo.anaconda.com/pkgs/msys2
Platform: win-64
Collecting package metadata (repodata.json): | |

```

Python 对象类型 (type), 包括字符串 (str)、字节串 (bytes)、整数 (int)、浮点数 (float)、布尔值 (bool)、列表 (list)、字典 (dict)、元组 (tuple)、集合 (set)

### 1. 字符串 ( str )

- 是什么: 表示文本, 用单引号 `' '` 或双引号 `" "` 包裹。
- 例子:

```
python
name = "Alice" # 字符串
greeting = '你好!' # 支持中文
```

### 2. 字节串 ( bytes )

- 是什么: 表示二进制数据 (如文件、网络传输), 前缀 `b`。
- 例子:

```
python
data = b'\x48\x65\x6c\x6c\x6f' # 字节串 (对应 ASCII 的 "Hello")
```

### 5. 布尔值 ( bool )

- 是什么: 表示逻辑值, 只有 `True` 或 `False`。
- 例子:

```
python
is_rainy = True # 布尔值
has_permission = False
```

### 6. 列表 ( list )

- 是什么: 可变的有序集合, 用方括号 `[]` 包裹。
- 例子:

```
python
fruits = ["苹果", "香蕉", "橘子"] # 列表
fruits.append("西瓜") # 可以修改
```

### 7. 字典 ( dict )

- 是什么: 存储键值对 ( `key:value` ), 用花括号 `{}` 包裹。
- 例子:

```
python
person = {"name": "Bob", "age": 30} # 字典
print(person["name"]) # 输出 "Bob"
```

### 8. 元组 ( tuple )

- 是什么: 不可变的有序集合, 用圆括号 `()` 包裹。
- 例子:

```
python
coordinates = (10.5, 20.3) # 元组
# coordinates[0] = 5 # 报错! 不能修改
```

### 9. 集合 ( set )

- 是什么: 无序且不重复的元素集合, 用花括号 `{}` (但无键值对)。
- 例子:

```
python
unique_numbers = {1, 2, 2, 3} # 自动去重 → {1, 2, 3}
```

### 对比总结:

类型	可变性	例子	用途
str	不可变	"Hello"	存储文本
bytes	不可变	b"ABC"	处理二进制数据
int	不可变	42	计算、计数
float	不可变	3.14	科学计算、测量
bool	不可变	True	逻辑判断
list	可变	[1, 2, 3]	动态数据集合
dict	可变	{"key": "value"}	键值对映射 (如 JSON)
tuple	不可变	(1, "a")	固定数据 (如坐标)
set	可变	{1, 2, 3}	去重、集合运算

### 小技巧: 查看类型

用 `type()` 函数检查对象的类型:

```
python
print(type("Hello")) # 输出 <class 'str'>
print(type([1, 2])) # 输出 <class 'list'>
```

```
use_of_int.py
1 # 整数创建
2 my_int = 10
3
4 # 整数运算
5 result = my_int + 5
6 print(result) # 输出 15
7
8 # 整数除法
9 div_result = my_int / 2
10 print(div_result) # 输出 5.0
11
12 # 整数取整除法
13 floor_div_result = my_int // 3
14 print(floor_div_result) # 输出 3
```

```
use_of_str.py
1 # 字符串创建
2 my_str = "Hello, World!"
3
4 # 字符串索引
5 print(my_str[0]) # 输出 'H'
6
7 # 字符串切片
8 print(my_str[0:5]) # 输出 'Hello'
9
10 # 字符串拼接
11 new_str = my_str + " How are you?"
12 print(new_str)
13
14 # 字符串方法
15 print(my_str.upper()) # 输出全大写字字符串
```

字符串创建：`my_str = "Hello, World!"` 这行代码创建了一个字符串对象 `my_str`，其值为 "Hello, World!"。

字符串索引：`print(my_str[0])` 利用索引来访问字符串中的单个字符。在 Python 里，字符串的索引从 0 开始，所以 `my_str[0]` 访问的是字符串的第一个字符，也就是 'H'。

字符串切片：`print(my_str[0:5])` 运用切片操作来获取字符串的一部分。切片操作的格式是 `[start:stop]`，`start` 是起始索引，`stop` 是结束索引（不包含该索引对应的字符）。因此，`my_str[0:5]` 会返回从索引 0 到 4 的字符，即 "Hello"。

字符串拼接：`new_str = my_str + " How are you?"` 通过 `+` 运算符把两个字符串连接起来，生成一个新的字符串 `new_str`。

字符串方法：`print(my_str.upper())` 调用了字符串对象的 `upper()` 方法，该方法会把字符串中的所有小写字母转换为大写字母，最后输出 "HELLO, WORLD!"。

整数创建：`my_int = 10` 创建了一个整数对象 `my_int`，其值为 10。

整数运算：`result = my_int + 5` 对整数进行加法运算，把 `my_int` 和 5 相加，结果存储在 `result` 中，最后输出 15。

整数除法：`div_result = my_int / 2` 使用 `/` 运算符进行除法运算，得到的结果是浮点数，所以输出为 5.0。

整数取整除法：`floor_div_result = my_int // 3` 运用 `//` 运算符进行取整除法，只返回商的整数部分，所以输出为 3。

```
use_of_list.py
1  # 列表创建
2  my_list = [1, 2, 3, 4, 5]
3
4  # 列表索引
5  print(my_list[2]) # 输出 3
6
7  # 列表切片
8  print(my_list[1:3]) # 输出 [2, 3]
9
10 # 列表追加元素
11 my_list.append(6)
12 print(my_list)
13
14 # 列表移除元素
15 my_list.remove(3)
16 print(my_list)
```

```
use_of_dict.py
1  # 字典创建
2  my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}
3
4  # 访问字典元素
5  print(my_dict['name']) # 输出 'John'
6
7  # 修改字典元素
8  my_dict['age'] = 31
9  print(my_dict)
10
11 # 添加新的键值对
12 my_dict['job'] = 'Engineer'
13 print(my_dict)
14
15 # 删除键值对
16 del my_dict['city']
17 print(my_dict)
```

列表创建：`my_list = [1, 2, 3, 4, 5]` 创建了一个包含 5 个整数元素的列表对象 `my_list`。

列表索引：`print(my_list[2])` 借助索引访问列表中的元素，列表索引同样从 0 开

始，所以 `my_list[2]` 访问的是列表的第三个元素，输出为 3。

列表切片：`print(my_list[1:3])` 利用切片操作获取列表的一部分，`my_list[1:3]` 返回从索引 1 到 2 的元素，即 `[2, 3]`。

列表追加元素：`my_list.append(6)` 调用列表的 `append()` 方法，在列表的末尾添加一个新元素 6。

列表移除元素：`my_list.remove(3)` 调用列表的 `remove()` 方法，从列表中移除值为 3 的元素。

字典创建：`my_dict = {'name': 'John', 'age': 30, 'city': 'New York'}` 创建了一个字典对象 `my_dict`，其中包含三个键值对。

访问字典元素：`print(my_dict['name'])` 通过键来访问字典中的值，`my_dict['name']` 返回与键 'name' 关联的值，即 'John'。

修改字典元素：`my_dict['age'] = 31` 通过键来修改字典中对应的值，将 'age' 键对应的值从 30 改为 31。

添加新的键值对：`my_dict['job'] = 'Engineer'` 为字典添加一个新的键值对，键为 'job'，值为 'Engineer'。

删除键值对：`del my_dict['city']` 使用 `del` 语句删除字典中指定键的键值对，这里删除了键为 'city' 的键值对。

```
# conda environments:
#
base                  * D:\anaconda
myproject             D:\anaconda\envs\myproject
prj1                  D:\anaconda\envs\prj1
prj2                  D:\anaconda\envs\prj2
week04                D:\anaconda\envs\week04
week05                D:\anaconda\envs\week05

(base) LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ conda activate week05
(week05)
LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ python use_of_ste.py
D:\anaconda\envs\week05\python.exe: can't open file 'C:\\Users\\
ek05\\use_of_ste.py': [Errno 2] No such file or directory
(week05)
LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
2551800159312
(week05)
```

# 创建不同类型的对象

```
num = 10
```

```
string = "Hello"
```

```
my_list = [1, 2, 3]
```

```
my_dict = {'key': 'value'}
```

# 使用 id() 函数返回对象在虚拟内存中的地址

```
print(f'整数对象 num 的内存地址: {id(num)}')
```

```
print(f'字符串对象 string 的内存地址: {id(string)}')
```

```
print(f'列表对象 my_list 的内存地址: {id(my_list)}')
```

```
print(f'字典对象 my_dict 的内存地址: {id(my_dict)}')
```

# 使用 type() 函数返回对象的类型

```
print(f'num 的类型: {type(num)}')
```

```
print(f'string 的类型: {type(string)}')
```

```
print(f'my_list 的类型: {type(my_list)}')
```

```
print(f'my_dict 的类型: {type(my_dict)}')
```

# 使用 isinstance() 函数判断对象是否属于某个 (或某些) 类型

```
print(f'num 是否为整数类型: {isinstance(num, int)}')
```

```
print(f'string 是否为字符串类型: {isinstance(string, str)}')
```

```
print(f'my_list 是否为列表类型: {isinstance(my_list, list)}')
```

```
print(f'my_dict 是否为字典类型: {isinstance(my_dict, dict)}')
```

# 使用 dir() 函数返回对象所支持的属性 (attributes) 的名称列表

```
print(f'整数对象 num 支持的属性: {dir(num)}')
```

```
print(f'字符串对象 string 支持的属性: {dir(string)}')
```

```
print(f'列表对象 my_list 支持的属性: {dir(my_list)}')
```

```
print(f'字典对象 my_dict 支持的属性: {dir(my_dict)}')
```

# 使用 str() 函数返回对象 print 时要显示在终端的字符串

```
print(f'整数对象 num 转换为字符串: {str(num)}')
```

```
print(f'列表对象 my_list 转换为字符串: {str(my_list)}')
```

# 使用 assert 语句查验某个表达式 (expression) 为真, 否则报错 (AssertionError) 退出

try:

```
    assert num > 5, "num 应该大于 5"
```

```
    print("assert 验证通过, num 大于 5")
```

except AssertionError as e:

```
    print(f'AssertionError: {e}')
```

# 使用 try 语句拦截报错, 避免退出, 将流程 (flow) 转入 except 语句

try:

```
    result = 1 / 0 # 会引发 ZeroDivisionError
```

except ZeroDivisionError:

```
    print("捕获到除零错误, 避免程序退出")
```

# 调用 breakpoint() 函数暂停程序运行, 进入 pdb 调试 (debug) 模式

# breakpoint()

# 若要使用调试功能, 可取消上面这行代码的注释, 运行程序时就会暂停, 进入 pdb 调试模式

结果:

整数对象 num 的内存地址: 139681032047184

字符串对象 string 的内存地址: 139680927334896

列表对象 my\_list 的内存地址: 139680927011776

字典对象 my\_dict 的内存地址: 139680927315968

num 的类型: <class 'int'>

string 的类型: <class 'str'>

my\_list 的类型: <class 'list'>

my\_dict 的类型: <class 'dict'>

num 是否为整数类型: True

string 是否为字符串类型: True

my\_list 是否为列表类型: True

my\_dict 是否为字典类型: True

整数对象 num 支持的属性: ['\_\_abs\_\_', '\_\_add\_\_', '\_\_and\_\_', '\_\_bool\_\_', '\_\_ceil\_\_',

'\_\_class\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_divmod\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_float\_\_',  
'\_\_floor\_\_', '\_\_floordiv\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattr\_\_', '\_\_getnewargs\_\_',  
'\_\_gt\_\_', '\_\_hash\_\_', '\_\_index\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_int\_\_', '\_\_invert\_\_',  
'\_\_le\_\_', '\_\_lshift\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_neg\_\_', '\_\_new\_\_',  
'\_\_or\_\_', '\_\_pos\_\_', '\_\_pow\_\_', '\_\_radd\_\_', '\_\_rand\_\_', '\_\_rdivmod\_\_', '\_\_reduce\_\_',  
'\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rfloordiv\_\_', '\_\_rlshift\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_',  
'\_\_ror\_\_', '\_\_round\_\_', '\_\_rpow\_\_', '\_\_rrshift\_\_', '\_\_rshift\_\_', '\_\_rsub\_\_', '\_\_rtruediv\_\_',  
'\_\_rxor\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_sub\_\_', '\_\_subclasshook\_\_',  
'\_\_truediv\_\_', '\_\_trunc\_\_', '\_\_xor\_\_', 'as\_integer\_ratio', 'bit\_length', 'conjugate',  
'denominator', 'from\_bytes', 'imag', 'numerator', 'real', 'to\_bytes']

字符串对象 string 支持的属性: ['\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_',  
'\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattr\_\_', '\_\_getitem\_\_',  
'\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_',  
'\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_',  
'\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_',  
'\_\_str\_\_', '\_\_subclasshook\_\_', 'capitalize', 'casefold', 'center', 'count', 'encode',  
'endswith', 'expandtabs', 'find', 'format', 'format\_map', 'index', 'isalnum', 'isalpha',  
'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',  
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix',  
'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',  
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']

列表对象 my\_list 支持的属性: ['\_\_add\_\_', '\_\_class\_\_', '\_\_class\_getitem\_\_',  
'\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_',  
'\_\_ge\_\_', '\_\_getattr\_\_', '\_\_getitem\_\_', '\_\_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']

字典对象 my\_dict 支持的属性: ['\_\_class\_\_', '\_\_class\_getitem\_\_', '\_\_contains\_\_',  
'\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_',  
'\_\_getattr\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_',  
'\_\_ior\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_or\_\_',  
'\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_reversed\_\_', '\_\_ror\_\_', '\_\_setattr\_\_',  
'\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'clear', 'copy', 'fromkeys',



'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']

整数对象 num 转换为字符串: 10

列表对象 my\_list 转换为字符串: [1, 2, 3]

assert 验证通过, num 大于 5

捕获到除零错误, 避免程序退出

```
! environment.yml U use_of_str.py U X use_of_bytes.py U
use_of_str.py > ...
1 a = [2, 5]
2 b = [2, 5]
3 x = id(a)
4 y = id(b)
5 print(x)
6 print(y)
7 a[0] = 9
8 print(a)
9 print(b)
10 print(id(a))
11 print(id(b))
12 print(type(a))
13 print(isinstance(a, str))
14 print(dir(a))
15
```

```
LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
1773379524864
1773379522880
[9, 5]
[2, 5]
1773379524864
1773379522880
<class 'list'>
False
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__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)
```

```
! environment.yml U X use_of_str.py U X use_of_bytes.py U
use_of_str.py > ...
1 a = [2, 5]
2 b = [2, 5]
3 x = id(a)
4 y = id(b)
5 print(x)
6 print(y)
7 a[0] = 9
8 print(a)
9 print(b)
10 print(id(a))
11 print(id(b))
12 print(type(a))
13 print(isinstance(a, str))
14 print(isinstance(a, list))
15 print(isinstance(a, (str, float)))
16 try:
17     assert isinstance(a, str)
18 except AssertionError:
19     print("type error")
20 print(["goodbye"])
21
```

```
LENOVO@LAPTOP-STLH7AK0 MINGW64 ~/repo/week05 (main)
$ python use_of_str.py
2834666428672
2834666426688
[9, 5]
[2, 5]
2834666428672
2834666426688
<class 'list'>
False
True
False
type error
goodbye
(week05)
```



# 1. int 类型

# 字面值

int\_literal = 10

print(f'字面值创建 int 实例: {int\_literal}')

# 初始化

int\_init = int(5)

print(f'初始化创建 int 实例: {int\_init}')

# 运算值

int\_operator = 2 + 3

print(f'运算值创建 int 实例: {int\_operator}')

# 2. float 类型

# 字面值

float\_literal = 3.14

print(f'字面值创建 float 实例: {float\_literal}')

# 初始化

float\_init = float(2)

print(f'初始化创建 float 实例: {float\_init}')

# 运算值

float\_operator = 1.5 \* 2

print(f'运算值创建 float 实例: {float\_operator}')

# 3. str 类型

# 字面值

str\_literal = "Hello"

print(f'字面值创建 str 实例: {str\_literal}')

# f-string 语法

```
name = "John"
str_f_string = f'Hello, {name}!'
print(f'f-string 语法创建 str 实例: {str_f_string}')
```

```
# 初始化
str_init = str(123)
print(f'初始化创建 str 实例: {str_init}')
```

```
# 运算值
str_operator = "Hello" + " World"
print(f'运算值创建 str 实例: {str_operator}')
```

#### # 4. list 类型

```
# 字面值
list_literal = [1, 2, 3]
print(f'字面值创建 list 实例: {list_literal}')
```

```
# 推导式
list_comprehension = [i * 2 for i in range(3)]
print(f'推导式创建 list 实例: {list_comprehension}')
```

```
# 初始化
list_init = list((4, 5, 6))
print(f'初始化创建 list 实例: {list_init}')
```

```
# 运算值
list_operator = list_literal + list_init
print(f'运算值创建 list 实例: {list_operator}')
```

```
# 索引值
list_index = list_operator[1]
print(f'索引值创建 (获取元素) 实例: {list_index}')
```

```
# 返回值
def return_list():
    return [7, 8, 9]

list_return = return_list()
print(f'返回值创建 list 实例: {list_return}')
```

```
# 5. dict 类型
# 字面值
dict_literal = {'name': 'John', 'age': 30}
print(f'字面值创建 dict 实例: {dict_literal}')
```

```
# 推导式
dict_comprehension = {i: i * 2 for i in range(3)}
print(f'推导式创建 dict 实例: {dict_comprehension}')
```

```
# 初始化
dict_init = dict([('city', 'New York'), ('job', 'Engineer')])
print(f'初始化创建 dict 实例: {dict_init}')
```

```
# 运算值
dict_new = dict_literal.copy()
dict_new.update(dict_init)
print(f'运算值创建 dict 实例: {dict_new}')
```

```
# 索引值
dict_index = dict_new['name']
print(f'索引值创建 (获取元素) 实例: {dict_index}')
```

```
# 返回值
def return_dict():
    return {'color': 'red'}
```

```
dict_return = return_dict()
print(f'返回值创建 dict 实例: {dict_return}')
```

# 6. set 类型

# 字面值

```
set_literal = {1, 2, 3}
print(f'字面值创建 set 实例: {set_literal}')
```

# 推导式

```
set_comprehension = {i * 2 for i in range(3)}
print(f'推导式创建 set 实例: {set_comprehension}')
```

# 初始化

```
set_init = set([4, 5, 6])
print(f'初始化创建 set 实例: {set_init}')
```

# 运算值

```
set_operator = set_literal.union(set_init)
print(f'运算值创建 set 实例: {set_operator}')
```

# 返回值

```
def return_set():
    return {7, 8, 9}
```

```
set_return = return_set()
print(f'返回值创建 set 实例: {set_return}')
```

结果:

```
运算值创建 str 实例: Hello World
字面值创建 list 实例: [1, 2, 3]
推导式创建 list 实例: [0, 2, 4]
初始化创建 list 实例: [4, 5, 6]
运算值创建 list 实例: [1, 2, 3, 4, 5, 6]
索引值创建 (获取元素) 实例: 2
返回值创建 list 实例: [7, 8, 9]
字面值创建 dict 实例: {'name': 'John', 'age': 30}
推导式创建 dict 实例: {0: 0, 1: 2, 2: 4}
初始化创建 dict 实例: {'city': 'New York', 'job': 'Engineer'}
运算值创建 dict 实例: {'name': 'John', 'age': 30, 'city': 'New York', 'job': 'Engineer'}
索引值创建 (获取元素) 实例: John
返回值创建 dict 实例: {'color': 'red'}
字面值创建 set 实例: {1, 2, 3}
推导式创建 set 实例: {0, 2, 4}
初始化创建 set 实例: {4, 5, 6}
运算值创建 set 实例: {1, 2, 3, 4, 5, 6}
返回值创建 set 实例: {8, 9, 7}
```

# 1. int 类型

```
num1 = 5
```

```
num2 = 3
```

# 数学运算符支持

```
print("int 数学运算: ")
```

```
print(f'{num1} + {num2} = {num1 + num2}')
```

```
print(f'{num1} - {num2} = {num1 - num2}')
```

```
print(f'{num1} * {num2} = {num1 * num2}')
```

```
print(f'{num1} ** {num2} = {num1 ** num2}')
```

```
print(f'{num1} / {num2} = {num1 / num2}')
```

```
print(f'{num1} // {num2} = {num1 // num2}')
```

```
print(f'{num1} % {num2} = {num1 % num2}')
```

# 判断相等

```
print(f'int 判断相等: {num1 == num2}')
```

# 比较运算符支持

```
print("int 比较运算: ")
```

```
print(f'{num1} > {num2}: {num1 > num2}')
```

```
print(f'{num1} < {num2}: {num1 < num2}')
```

```
print(f'{num1} >= {num2}: {num1 >= num2}')
```

```
print(f'{num1} <= {num2}: {num1 <= num2}')
```

# 布尔值映射

```
print(f'int 布尔值映射: bool({num1}) = {bool(num1)}, bool(0) = {bool(0)}')
```

# 不可迭代

```
try:
```

```
    for i in num1:
```

```
        print(i)
```

```
except TypeError:
```

```
    print("int 不可迭代")
```

```
# 不支持返回长度
try:
    print(len(num1))
except TypeError:
    print("int 不支持返回长度")
```

```
# 不支持索引操作
try:
    print(num1[0])
except TypeError:
    print("int 不支持索引操作")
```

```
# 常用方法
print(f'int 常用方法: {dir(int)}")
```

# 2. float 类型

```
float1 = 5.0
```

```
float2 = 3.0
```

```
# 数学运算符支持
print("\nfloat 数学运算: ")
print(f'{float1} + {float2} = {float1 + float2}')
print(f'{float1} - {float2} = {float1 - float2}')
print(f'{float1} * {float2} = {float1 * float2}')
print(f'{float1} ** {float2} = {float1 ** float2}')
print(f'{float1} / {float2} = {float1 / float2}')
print(f'{float1} // {float2} = {float1 // float2}')
print(f'{float1} % {float2} = {float1 % float2}')
```

```
# 判断相等
print(f'float 判断相等: {float1 == float2}')
```

# 比较运算符支持

```
print("float 比较运算: ")
print(f'{float1} > {float2}: {float1 > float2}')
print(f'{float1} < {float2}: {float1 < float2}')
print(f'{float1} >= {float2}: {float1 >= float2}')
print(f'{float1} <= {float2}: {float1 <= float2}')
```

# 布尔值映射

```
print(f'float 布尔值映射: bool({float1}) = {bool(float1)}, bool(0.0) = {bool(0.0)}')
```

# 不可迭代

```
try:
    for i in float1:
        print(i)
except TypeError:
    print("float 不可迭代")
```

# 不支持返回长度

```
try:
    print(len(float1))
except TypeError:
    print("float 不支持返回长度")
```

# 不支持索引操作

```
try:
    print(float1[0])
except TypeError:
    print("float 不支持索引操作")
```

# 常用方法

```
print(f'float 常用方法: {dir(float)}')
```

# 3. str 类型



```
str1 = "hello"
str2 = "world"
```

```
# 数学运算符支持
```

```
print("\nstr 数学运算: ")
print(f'{str1} + {str2} = {str1 + str2}')
print(f'{str1} * 3 = {str1 * 3}')
```

```
# 判断相等
```

```
print(f'str 判断相等: {str1 == str2}')
```

```
# 比较运算符支持
```

```
print("str 比较运算: ")
print(f'{str1} > {str2}: {str1 > str2}')
print(f'{str1} < {str2}: {str1 < str2}')
print(f'{str1} >= {str2}: {str1 >= str2}')
print(f'{str1} <= {str2}: {str1 <= str2}')
```

```
# 布尔值映射
```

```
print(f'str 布尔值映射: bool({str1}) = {bool(str1)}, bool("") = {bool("")}')
```

```
# 可迭代
```

```
print("str 迭代: ")
for char in str1:
    print(char)
```

```
# 支持返回长度
```

```
print(f'str 长度: {len(str1)}')
```

```
# 支持索引操作
```

```
print(f'str 索引操作: {str1[0]}')
```

```
# 常用方法
```

```
print(f'str 常用方法: {dir(str)}')
```

# 4. list 类型

```
list1 = [1, 2, 3]
```

```
list2 = [4, 5, 6]
```

# 数学运算符支持

```
print("\nlist 数学运算: ")
```

```
print(f'{list1} + {list2} = {list1 + list2}')
```

```
print(f'{list1} * 2 = {list1 * 2}')
```

# 判断相等

```
print(f'list 判断相等: {list1 == list2}')
```

# 比较运算符支持

```
print("list 比较运算: ")
```

```
print(f'{list1} > {list2}: {list1 > list2}')
```

```
print(f'{list1} < {list2}: {list1 < list2}')
```

```
print(f'{list1} >= {list2}: {list1 >= list2}')
```

```
print(f'{list1} <= {list2}: {list1 <= list2}')
```

# 布尔值映射

```
print(f'list 布尔值映射: bool({list1}) = {bool(list1)}, bool([]) = {bool([])}')
```

# 可迭代

```
print("list 迭代: ")
```

```
for item in list1:
```

```
    print(item)
```

# 支持返回长度

```
print(f'list 长度: {len(list1)}')
```

# 支持索引操作

```
print(f'list 索引操作: {list1[0]}')
```

# 常用方法

```
print(f'list 常用方法: {dir(list)}')
```

# 5. dict 类型

```
dict1 = {'a': 1, 'b': 2}
```

```
dict2 = {'c': 3, 'd': 4}
```

# 不支持大部分数学运算符

try:

```
    print(dict1 + dict2)
```

except TypeError:

```
    print("\ndict 不支持 + 运算符")
```

# 判断相等

```
print(f'dict 判断相等: {dict1 == dict2}')
```

# 比较运算符不支持 (Python 3 中)

try:

```
    print(dict1 > dict2)
```

except TypeError:

```
    print("dict 不支持比较运算符")
```

# 布尔值映射

```
print(f'dict 布尔值映射: bool({dict1}) = {bool(dict1)}, bool({}) = {bool({})}')
```

# 可迭代

```
print("dict 迭代: ")
```

```
for key in dict1:
```

```
    print(key)
```

# 支持返回长度

```
print(f'dict 长度: {len(dict1)}')
```

```
# 支持索引操作（通过键）
print(f'dict 索引操作: {dict1['a']}')
```

```
# 常用方法
print(f'dict 常用方法: {dir(dict)}')
```

## # 6. set 类型

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
```

```
# 数学运算符支持（部分）
print("\nset 数学运算: ")
print(f'{set1} | {set2} = {set1 | set2}') # 并集
print(f'{set1} & {set2} = {set1 & set2}') # 交集
print(f'{set1} - {set2} = {set1 - set2}') # 差集
```

- 这部分代码测试了集合类型对部分数学运算符的支持：
  - `|` 用于求两个集合的并集，即包含两个集合中所有元素的集合。
  - `&` 用于求两个集合的交集，即包含两个集合中共同元素的集合。
  - `-` 用于求两个集合的差集，即包含在第一个集合中但不在第二个集合中的元素的集合。

```
# 判断相等
print(f'set 判断相等: {set1 == set2}')
```

```
# 比较运算符支持（部分）
print("set 比较运算: ")
print(f'{set1} > {set2}: {set1 > set2}') # 真超集判断
print(f'{set1} < {set2}: {set1 < set2}') # 真子集判断
print(f'{set1} >= {set2}: {set1 >= set2}') # 超集判断
print(f'{set1} <= {set2}: {set1 <= set2}') # 子集判断
```

```
# 布尔值映射
print(f'set 布尔值映射: bool({set1}) = {bool(set1)}, bool(set()) = {bool(set())}')
```

```
# 可迭代
print("set 迭代: ")
for item in set1:
    print(item)

# 支持返回长度
print(f"set 长度: {len(set1)}")

# 不支持索引操作
try:
    print(set1[0])
except TypeError:
    print("set 不支持索引操作")

# 常用方法
print(f"set 常用方法: {dir(set)}")
```

结果:

```
int 数学运算:
5 + 3 = 8
5 - 3 = 2
5 * 3 = 15
5 ** 3 = 125
5 / 3 = 1.6666666666666667
5 // 3 = 1
5 % 3 = 2
int 判断相等: False
int 比较运算:
5 > 3: True
5 < 3: False
5 >= 3: True
5 <= 3: False
int 布尔值映射: bool(5) = True, bool(0) = False
int 不可迭代
int 不支持返回长度
int 不支持索引操作
int 常用方法: ['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
```

```
float 数学运算:
5.0 + 3.0 = 8.0
5.0 - 3.0 = 2.0
5.0 * 3.0 = 15.0
5.0 ** 3.0 = 125.0
5.0 / 3.0 = 1.6666666666666667
5.0 // 3.0 = 1.0
5.0 % 3.0 = 2.0
float 判断相等: False
float 比较运算:
5.0 > 3.0: True
5.0 < 3.0: False
5.0 >= 3.0: True
5.0 <= 3.0: False
float 布尔值映射: bool(5.0) = True, bool(0.0) = False
float 不可迭代
float 不支持返回长度
float 不支持索引操作
float 常用方法: ['__abs__', '__add__', '__bool__', '__class__', '__delattr__', '__di
```

```
str 数学运算:
hello + world = helloworld
hello * 3 = hellohellohello
str 判断相等: False
str 比较运算:
hello > world: False
hello < world: True
hello >= world: False
hello <= world: True
str 布尔值映射: bool(hello) = True, bool('') = False
str 迭代:
h
e
l
l
o
str 长度: 5
str 索引操作: h
str 常用方法: ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__
```

```
list 数学运算:
[1, 2, 3] + [4, 5, 6] = [1, 2, 3, 4, 5, 6]
[1, 2, 3] * 2 = [1, 2, 3, 1, 2, 3]
list 判断相等: False
list 比较运算:
[1, 2, 3] > [4, 5, 6]: False
[1, 2, 3] < [4, 5, 6]: True
[1, 2, 3] >= [4, 5, 6]: False
[1, 2, 3] <= [4, 5, 6]: True
list 布尔值映射: bool([1, 2, 3]) = True, bool([]) = False
list 迭代:
1
2
3
list 长度: 3
list 索引操作: 1
list 常用方法: ['__add__', '__class__', '__contains__', '__delattr__', '__delitem__',
```

```
dict 不支持 + 运算符
dict 判断相等: False
dict 不支持比较运算符
dict 布尔值映射: bool({'a': 1, 'b': 2}) = True, bool({}) = False
dict 迭代:
a
b
dict 长度: 2
dict 索引操作: 1
dict 常用方法: ['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
```

```
set 数学运算:
{1, 2, 3} | {5, 4, 5} = {1, 2, 3, 4, 5}
{1, 2, 3} & {5, 4, 5} = {3}
{1, 2, 3} - {5, 4, 5} = {1, 2}
set 判断相等: False
set 比较运算:
{1, 2, 3} > {3, 4, 5}: False
{1, 2, 3} < {5, 4, 5}: False
{1, 2, 3} >= {3, 4, 5}: False
{1, 2, 3} <= {5, 4, 5}: False
set 布尔值映射: bool({1, 2, 3}) = True, bool(set()) = False
set 迭代:
1
2
3
set 长度: 3
set 索引操作
set 常用方法: ['__and__', '__class__', '__contains__', '__delattr__', '__dir__', '__
```