上周 (初级) 我们讲过 *Python 编程本质上是拼接操纵各种对象*，因而在本周，我们的目标是要掌握最基础、最常用的几种 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`)。这几种类型都是 Python 解释器 **内置的** (built-in)，不需要任何导入 (import)。

另外，Python 标准库 (standard library) 里的 `pathlib` 和 `datetime` 模块 (module) 提供了用于处理 **路径** 和 **日期时间** 的类型，也是非常基础、非常常用的。标准库模块都不需要安装 (pip/conda install)，但使用前需要导入 (import)。

1. 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
2. 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
3. *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`：
   - 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)
   - 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 (正整数)，如果 `id(a) == id(b)`，那么 `a is b` (`is` 是个运算符)
     - `type()` -- 返回对象的类型
     - `isinstance()` -- 判断对象是否属于某个 (或某些) 类型
     - `dir()` -- 返回对象所支持的属性 (attributes) 的名称列表
     - `str()` -- 返回对象 `print` 时要显示在终端的字符串
   - 可以调用 `print()` 函数将表达式 (expression) 输出到终端，查看结果是否符合预期
   - 可以利用 `assert` 语句查验某个表达式 (expression) 为真，否则报错 (`AssertionError`) 退出
   - 可以利用 `try` 语句拦截报错，避免退出，将流程 (flow) 转入 `except` 语句
   - 可以调用 `breakpoint()` 函数暂停程序运行，进入 `pdb` 调试 (debug) 模式
4. 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)
5. 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)

   建议先在 `pdb` 里试验，然后把确定能够运行的代码写在 `use_of_{name}.py` 文件里
6. 将你学习理解实践这些概念所产生的笔记，以及运行用的 `.py` 代码，都 `add` 、 `commit` 、 `push` 到 GitCode 平台你名下的仓库里，最后提交 PR

```python
a = "hello"
x = id(a)
print(x)
```

```
Think@LAPTOP-IVVORB8S MINGW64 ~/chao/week05 (main)
$ python use_of_str.py
2193048755328
(week05)
Think@LAPTOP-IVVORB8S MINGW64 ~/chao/week05 (main)
$ python use_of_str.py
2564707146880
(week05)
```

Id 用于显示地址

use_of_str.py > ...

```python
a = "hello"
b = "hello"
x = id(a)
y = id(b)
print(x)
print(y)

```

```
Think@LAPTOP-IVVORB8S MINGW64 ~/chao/week05 (main)
$ python use_of_str.py
2078497004672
2078497004672
(week05)
```

Id 相同

use_of_str.py > ...

tracked

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

```

```
Think@LAPTOP-IVVORB8S MINGW64 ~/chao/week05
$ python use_of_str.py
1797588982016
1797588980032
(week05)
```

虽然 a 和 b 都是完全相等的值，但是 a 和 b 是完全两个不同的对象，所以 id 不同

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
```

结果是相同的，遇到不确定可以自己运行一下

```
Think@LAPTOP-IVVORB8S MINGW64 ~/chao/week05 (main)
$ python use_of_str.py
1547686254848
1547686252864
[9, 5]
[2, 5]
1547686254848
1547686252864
(week05)
```

Instance 运用

```
$ python use_of_str.py
3086554241280
3086554239296
[9, 5]
[2, 5]
3086554241280
3086554239296
<class 'list'>
isinstance(a,str) False
(week05)
```

```
5   # print("isinstance(a,(str,float))")
6   try:
7       assert isinstance(a, str)
8   except AssertionError:
9       print("type error")
0   print("goodbye")
1
```

Try 可以拦截报错

```
$ python use_of_str.py
2192979073280
2192979071296
[9, 5]
[2, 5]
2192979073280
2192979071296
<class 'list'>
isinstance(a,str) False
type error
goodbye
(week05)
```

字符串

```
assert str([5, 8, 2]) == "[5,8,2]"
assert str([1.1 + 2.2]) == "3.3"
```

```
[5, 8, 2]
Traceback (most recent call last):
  File "C:\Users\Think\chao\week05\use_of_str.py", line 30, in <module>
    assert str([5, 8, 2]) == "[5,8,2]"
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^
AssertionError
(base)
```

出现报错，用 pdb 检查

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

在计算机二进制的保存下，计算出来并不是整数

Python 中的字符串无法修改，但是可以用 fomat 或者 upper 等命令去改变。

```
Traceback (most recent call last):
  File "C:\Users\Think\chao\week05\use_of_str.py", line 51, in <module>
    print(s2 - s1)
          ~~~^~~~
TypeError: unsupported operand type(s) for -: 'str' and 'str'
(base)
```

字符串不支持减和除法的格式，try 语句可以修改。

```
try:
    print(s2 - s1)
except TypeError as e:
    print(e)
```

```
raceback (most recent call last):
 File "C:\Users\Think\chao\week05\use_of_str.py", line 51, in <module>
   print(s2 - s1)
         ~~~^~~~
ypeError: unsupported operand type(s) for -: 'str' and 'str'
```

Try 语句可以让命令不报错，并且显示错误的信息

ASCII table ： 字符的排序规则

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

字符串可以做循环，把字符串里面都跑一遍

字节串

use_of_bytes.py > ...

```
1  s = b"hello"
2  print(s)
3  print(s[0])
4
```

序号中保存的是字节

```
$ python use_of_bytes.py
b'hello'
104
(week05)
```

Ascii 编码只能对标准的美国字符编码，不能对中国文字进行编码

```
(Pdb) p p.exists()
False
```

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

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

```
(Pdb) p p.exists()
True
```

因为 windows 和 unix 不兼容，修改一下就显示存在了

```
Think@LAPTOP-IVVORB8S MINGW64 ~
$ ls -l /d/anaconda/envs/week05/python
-rwxr-xr-x 5 Think 197121 93184  3月   5 06:39 /d/anaconda/envs/week05/python*
(week05)
Think@LAPTOP-IVVORB8S MINGW64 ~
$ ls -lh /d/anaconda/envs/week05/python
-rwxr-xr-x 5 Think 197121 91K  3月   5 06:39 /d/anaconda/envs/week05/python*
(week05)
Think@LAPTOP-IVVORB8S MINGW64 ~
$
```

1k 等于 1024 个字符

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

可以互相转换，二进制

编解码器不同，编出来的代码也不同