第 5 周 Python 对象类型 (初级)

任务目标

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

✅ **步骤 1：打开项目目录**

- 打开 **VS Code**。
- 点击左侧菜单中的“文件资源管理器”（或按快捷键 Ctrl + Shift + F）。
- 点击顶部菜单栏的 **File > Open Folder**（文件 > 打开文件夹）。
- 选择你要操作的项目目录并打开。

✅ **步骤 2：新建 environment.yml 文件**

- 在 VS Code 的左侧文件资源管理器中，右键点击项目根目录。
- 选择 **New File**（新建文件）。
- 输入文件名为：environment.yml，然后按回车。

✅ **步骤 3：编辑 environment.yml 内容**

在打开的 environment.yml 文件中输入以下内容来指定 Python 3.12 和环境名称（例如 myenv）：

yaml

步骤 4：保存文件

- 按下 Ctrl + S 或点击菜单栏的 **File > Save** 来保存文件。

✅ **步骤 5：运行终端命令创建环境**

- 打开 VS Code 的终端：
  - 点击顶部菜单栏的 **Terminal > New Terminal**（终端 > 新建终端）。
- 在终端中运行以下命令来创建 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

use_of_str.py

```
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(b))  # is is same as y?
12  print(type(a))
13  print('isinstance(a, str):', isinstance(a, str))
14  print('isinstance(a, list):', isinstance(a, list))
15  print(isinstance(a, (str, float)))
16  try:
17      assert isinstance(a, str)
18  except AssertionError:
19      breakpoint()
20      print('type error')
21  print('goodbye')
```

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)
    - 提取值 (subscription)
    - 返回值 (return value of function/method call)

```python
print('字面值')
s = 'university'
print(s)
print(isinstance(s, str))
assert type(s) is str

print('f-string')
x = 'Tom'
s = f'name: {x}'
print(s)

s = 'a\tb'
print('TAB', s)

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

```python
s = """xyz
abc
  eee
aaa
"""
print(s)

print('初始化')
s = str()
print(s)
s = str([5, 8, 2])
print(s)

assert str([5, 8, 2]) == '[5, 8, 2]'
assert str(1.1 + 2.2) != '3.3'

assert str() == ''

s = '='
x = id(s)
s = s * 20
y = id(s)
print(s)
assert x != y

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

```python
assert s[3] == 'l'
assert s[-1] == 'o'
assert s[:3] == 'hel'
assert s[4] == s[-1]
try:
    s[5]
except IndexError as e:
    print(e)

s = 'hello'
u = s.upper()
print(u)
print(s)

t = 'name: {}, age {}'
print(t)
t1 = t.format('Jack', 21)
print(t1)

s1 = 'abc'
s2 = 'ghi'
s = s1 + s2
assert s == 'abcghi'
print(s2 + s1)

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

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) 支持提取操作 (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