第五周学习报告

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

用 vscode 打开 week05 environment.yml 文件夹，改 04 为 05
然后在 week05 下运行 `conda env create` 命令

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

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)。

1.str

```python
    # 实例(字面值)
    v = "aaaa"
assert type(v) is str

print("f-string")
y1 = "honghong"
t1 = f"name: {y1}"
print(t1)

t = "c\td"
print("TAB", t)

# 初始化
print("chushihua")
t = str()
print(t)
t = str([95588, 8888, 123])
print(t)

# 运算值
t = "-"
t = t * 20
print(t)

# 索引值
t = "banana"
assert t[2] == "n"  # python的字符串是从0开始计算
print(t[:4])

# 返回值
t = "tianqiyubao"
e = t.upper()
print(e)
```

```python

# 运算符号
i = "wangzhan"
j = "8901"
print(i + j)

t = "###-###"
t = t * 9
print(t)

assert "bbbcd" > "BBBCD"
print("def" > "456")

t = "tianqiyubao"
for char in t:
    print(char)
print(len(t))

t = "xiyouji:baimawen:sunwukong"
print(t.split(":"))

breakpoint()
```

2.bytes

use_of_bytes.py > ...

```python
from pathlib import Path

s = "你好，世界"
print(s)

p = Path("D:\\ananconda3\\envs\\week05")
s = p.read_bytes()
print(len(s))

p = Path("environment.yml")
b = p.read_bytes()
print(b[0])

# 字符串编码为字节
s_encoded = s.encode('utf-8')
print(s_encoded)

# 字节解码为字符串
s_decoded = s_encoded.decode('utf-8')
print(s_decoded)

# 断言检查类型
assert isinstance(s, str)
assert isinstance(s_encoded, bytes)

# 包含特殊字符的字符串操作
s_with_special = "abc你好😀"
print(s_with_special)
b_special = s_with_special.encode('utf-8')
print(b_special)

breakpoint()
```

3.int

use_of_int.py > a

```python
a = 42
b = 5
c = 7
# 加法运算
result_sum = a + b
print(f"{a} + {b} 的结果是: {result_sum}")
# 除法运算并断言结果
d = 21
e = 7
assert d / e == 3, "除法运算结果不符合预期"
# 取余运算并断言结果
f = 17
g = 5
assert f % g == 2, "取余运算结果不符合预期"

# 使用try - except捕获断言错误
try:
    # 故意设置一个错误的断言
    assert 0
except AssertionError as e:
    print(f"捕获到断言错误: {type(e)}")

# 定义一个整数并进入调试模式
num = 666

breakpoint()
```

4.float

```python
# 通过字符串转换为浮点数并检查类型
y = float('2.718')
print(type(y))

# 断言两个浮点数相等
assert x == y

x = 5.0
y = 2.0
print(x - y, type(x - y))

# 生成随机浮点数
x = random.random()
print(x)

# 检查浮点数是否为0
assert not 0.0

# 处理特殊浮点数NaN（非数字）
nan = float('nan')
print(nan + 5)
print(nan > 5)
print(nan < 5)
print(nan == 5)

# 处理正无穷大
pinf = float('inf')
print(pinf - 1020)
print(pinf > 1020)
print(pinf == pinf)

# 处理负无穷大
ninf = float('-inf')
print(ninf + 10)
print(ninf < -10)
print(ninf == ninf)
```

5. `bool`

```python
# 定义布尔变量
t = True
f = False

# 打印布尔值
print(f"定义的布尔值: t = {t}, f = {f}")

# 打印布尔值类型
print(f"t的类型: {type(t)}")
print(f"f的类型: {type(f)}")

breakpoint()
```

6. `list`

use_of_list.py > ...

```python
l = [1, 5, "abc"]
print(l)

try:
    print(l[0])  # 打印第一个元素
    print(l[1])  # 打印第二个元素
    print(l[2])  # 打印第三个元素
    print(l[3])  # 故意越界，触发IndexError
except IndexError as e:
    print("list index out of range")

# 切片操作
print(l[:2])  # 取前两个元素
print(l[1:])  # 取从第二个元素到末尾的元素

# 列表拼接
a = [2, 5]
b = ['a', 'c']
print(a + b)  # 拼接两个列表
print(b + a)  # 改变顺序

# 检查列表拼接是否满足交换律（实际不满足）
print(a + b == b + a)

# 尝试列表相减（会触发TypeError，因为列表不支持减法运算）
try:
    print(a - b)
except TypeError as e:
    print("unsupported operand type(s) for -: 'list' and 'list'")

# 列表重复
a = [2, 5]
print(a * 3)  # 列表重复3次

# 元素修改
a = [2, 5]
b = a * 3
b[0] = 9  # 修改第一个元素
print(b)

# 遍历列表
a = [2, 5, 8]
for i in a:
    print(i)

# 列表推导式
a = [2, 5, 8]
b = [i * 2 for i in a if i < 8]  # 对小于8的元素乘以2
print(b)

# 使用列表方法 - append
a = [2, 5]
x = a.append(4)  # append方法返回值为None，这里只是展示调用
print(x)
print(a)

# 使用列表方法 - extend
a = [2, 5]
a.extend([7, 9])  # 在列表末尾添加多个元素
print(a)

# 使用列表方法 - pop
a = [2, 5, 7]
popped = a.pop()  # 弹出并返回最后一个元素
print(popped)
print(a)
```

7. dict

use_of_dict.py > ...

```python
d = {'a': 1, 'bb': 5, 'cat': 3}
print(d)
# 打印输出字典d的类型
print(type(d))

# 遍历字典d，这里a代表字典的键，依次打印出字典的键
for a in d:
    print(a)

# 利用字典的keys()方法遍历字典d的键，和上面直接遍历字典效果相同，依次打印出字典的键
for a in d.keys():
    print(a)

# 利用字典的values()方法遍历字典d的值，依次打印出字典的值
for a in d.values():
    print(a)

# 使用列表推导式，将字典d的键值对以元组形式提取出来组成列表，赋值给变量l
l = [a for a in d.items()]
# 打印输出列表l
print(l)

# 遍历字典d的键值对，k代表键，v代表值，依次打印出键和值
for k, v in d.items():
    print(k, v)

breakpoint()
```

8. tuple

```python
print(t[0])
# 访问元组t的第二个元素并打印
print(t[1])
# 访问元组t的第三个元素并打印
print(t[2])

# 尝试修改元组t的第一个元素，元组是不可变类型，会触发TypeError
try:
    t[0] = 9
except TypeError as e:
    print(e)

# 创建一个空字典d
d = {}
# 向字典d中添加键值对，键为'abc'，值为5
d['abc'] = 5
# 向字典d中添加键值对，键为7，值为100
d[7] = 100

# 创建一个列表q
q = [3, 1]
# 尝试将列表q作为键添加到字典d中，列表是可变类型，不可哈希，会触发TypeError
try:
    d[q] = 21
except TypeError as e:
    print(e)

# 创建一个元组t，包含三个元素：整数100、元组(3)
t = (100, (3))
# 向字典d中添加键值对，键为元组t，值为21
d[t] = 21
# 打印字典d的内容
print(d)

# 访问字典d中键为(3, 1)的值并打印
print(d[(3, 1)])
```

9. set

use_of_set.py > ...

```python
s = {3, 5, 8}
print(s)
print(type(s))

# 尝试创建一个包含列表的集合，由于列表是可变类型，不能作为集合元素，会触发TypeError
try:
    s = {3, [5], 8}
except TypeError as e:
    print(e)

# 创建另一个集合q
q = {3, 4, 6, 4, 7, 6}
print(q)
print(type(q))

# 再次创建集合s
s = {6, 4, 7, 4, 4, 7}
print(s)
# 检查整数4是否在集合s中，返回True或False
print(4 in s)
# 检查集合s是否是自身的子集，返回True或False
print(s <= s)

# 创建集合s2
s2 = {7, 4, 7}
print(s2)
# 求集合s和s2的并集，返回一个新集合
print(s | s2)
# 检查集合s和s2是否相等，返回True或False
print(s == s2)
```

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.path

use_of_path.py > ...

```python
from pathlib import Path
from pprint import pprint

# 获取当前路径并创建Path对象
p = Path('.')
# 打印当前路径
print(p)
# 检查当前路径是否存在
print(p.exists
# 获取当前路径的
print(p.absolu
# 以美观的格式打
pprint(list(p.iterdir()))

# 获取当前路径下名为'data2'的子路径并创建Path对象
p = Path('./data2')
# 检查该路径是否存在
print(p.exists())
# 检查该路径是否为目录
print(p.is_dir())

# 获取当前路径
p = Path('.')
# 拼接当前路径和'README.txt'文件名，创建新的Path对象
p2 = p / 'README.txt'
# 获取p2的绝对路径
p3 = p2.absolute()
# 打印p2
print(p2)
# 打印p3
print(p3)
```

```
(method) def iterdir() -> Generator[Path, None, None]
Iterate over the files in this directory. Does not yield any
result for the special paths '.' and '..'.
```

2. datetime

use_of_datetime.py > ...

```python
from datetime import date, datetime, timedelta

# 获取当前日期
t1 = date.today()
print(t1)

# 创建指定日期对象
t2 = date(2025, 12, 25)
print(t2)

# 计算两个日期的时间差
td = t2 - t1
print(td)
print(type(td))
# 打印时间差的天数
print(td.days)

# 定义日期时间格式字符串
s = "2024-08-15"
# 将字符串解析为datetime对象
d1 = datetime.strptime(s, "%Y-%m-%d")
print(d1)

# 将datetime对象格式化为字符串
s2 = datetime.strftime(d1, "%Y-%m-%d")
print(s2)

```