Week05 帅占烜学习笔记

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)。

```python
# Part 1: 常用的对象检视函数和语句
# 使用 id、type、isinstance 和 dir 函数
num = 10
print(f"对象的 ID: {id(num)}")
print(f"对象的类型: {type(num)}")
print(f"对象是否为整数类型: {isinstance(num, int)}")
print(f"对象的所有属性和方法: {dir(num)}")

# 使用 assert 语句和 try-except 语句进行代码调试
try:
    assert num > 5, "num 应该大于 5"
    print("断言通过")
except AssertionError as e:
    print(e)


# Part 2: 获得 str 类型实例的几种途径
# 字符串字面值
str_literal_single = 'Hello, World!'
str_literal_double = "Hello, World!"
```

```python

# 推导式初始化
str_from_list = ''.join([chr(i) for i in range(65, 70)])

# 运算值
str1 = "Hello"
str2 = " World"
str_result = str1 + str2

# 索引值
str_index = str_result[0]

# 使用 assert 进行代码验证
assert isinstance(str_literal_single, str), "str_literal_single 应该是字符串类型"


# Part 3: str 类型支持的各种操作和方法
# 字符串运算
str_add = "Hello" + " World"
str_mul = "Hello" * 3
```

```python
assert "Hello" == "Hello", "两个字符串应该相等"

# 字符编码
ascii_char = ord('A')
char_from_ascii = chr(ascii_char)

# 排序规则
str_list = ['a', 'A', '1']
str_list.sort()

# translate 和 make_translation 函数
translation_table = str.maketrans('abc', '123')
translated_str = "abc".translate(translation_table)

# 常用方法
capitalized_str = "hello".capitalize()
centered_str = "hello".center(10)
count_char = "hello".count('l')

# 内置函数判断
```

```python
large_int = 12345678901234567890
bytes_needed = large_int.bit_length() // 8 + 1

# bytes 编解码
str_to_bytes = "Hello".encode('utf-8')
bytes_to_str = str_to_bytes.decode('utf-8')


# Part 5: float ~ dict 等类型
# 浮点数
float_num = 3.14
float_from_str = float("3.14")

# 布尔值
bool_value = True
bool_as_int = int(bool_value)

# 列表
list_num = [1, 2, 3]
list_operation = list_num + [4, 5]
```

python_types_demo.py

```python
dict_example = {'key1': 'value1', 'key2': 'value2'}


# Part 6: tuple ~ date 等类型
# 元组和列表的区别
tuple_example = (1, 2, 3)
list_example = [1, 2, 3]

# 不可修改的元组作为字典的键
dict_with_tuple_key = {(1, 2): 'value'}

# 集合运算
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1.union(set2)
intersection_set = set1.intersection(set2)
symmetric_difference_set = set1.symmetric_difference(set2)

# 日期时间
from datetime import datetime, date, time
```

```python
dict_with_tuple_key = {(1, 2): 'value'}

# 集合运算
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1.union(set2)
intersection_set = set1.intersection(set2)
symmetric_difference_set = set1.symmetric_difference(set2)

# 日期时间
from datetime import datetime, date, time

now = datetime.now()
formatted_date = now.strftime("%Y-%m-%d %H:%M:%S")
parsed_date = datetime.strptime("2024-01-01 12:00:00", "%Y-%m-%d %H:%M:%S")
```

控制台

- 对象的 ID: 140125548771920
- 对象的类型: <class 'int'>
- 对象是否为整数类型: True
- 对象的所有属性和方法: ['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__r... rmod__',

Here’s a simple example of how to define and use an abstract base class in Python:

Python animals.py

```python
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass

    def jump(self):
        return f"{self.__class__.__name__} is jumping"
```

Python

```python
>>> from animals import Animal

>>> class Dog(Animal): pass
...

>>> Dog()
Traceback (most recent call last):
    ...
TypeError: Can't instantiate abstract class Dog
    without an implementation for abstract method 'speak'
```

Here's a correct implementation:

Python

```python
>>> class Dog(Animal):
...     def speak(self):
...         return "Woof Woof"
...

>>> dog = Dog()
>>> dog.speak()
'Woof Woof'
```

```python
>>> class AsyncContextManager:
...     async def __aenter__(self):
...         print("Entering context: Setup logic here...")
...         return self
...     async def __aexit__(self, exc_type, exc_val, exc_tb):
...         print("Exiting context: Teardown logic here...")
...

>>> async def main():
...     async with AsyncContextManager():
...         print("Inside context: Your async code here...")
...

>>> import asyncio
>>> asyncio.run(main())
Entering context: Setup logic here...
Inside context: Your async code here...
Exiting context: Teardown logic here...
```

)

`range` Real-World Example

Say that you want to generate a sequence of numbers representing the leap years between 2000 and 2100:

Python

```python
>>> leap_years = range(2000, 2100, 4)
>>> list(leap_years)
[
    2000, 2004, 2008, 2012, 2016,
    2020, 2024, 2028, 2032, 2036,
    2040, 2044, 2048, 2052, 2056,
    2060, 2064, 2068, 2072, 2076,
    2080, 2084, 2088, 2092, 2096
]
```

In this example, the `range` object helps efficiently generate a list of leap years by specifying the `start`, `stop`, and `step` values.