

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`)。

阅读了包括以下及更多的学习资料

Python 词汇表 /

抽象基类（ABC）
注解
args（参数）
论点
数组
分配
异步上下文管理器
异步生成器
异步生成器迭代器 异步生成器迭代器
asynchronous iterable
异步迭代
异步迭代器
异步编程

Python 的内置数据类型

Python 的**内置数据类型**是用于存储和作不同类型数据的基本构造。它们有不同的用途，并具有与之关联的特定作。

以下是 Python 中关键内置数据类型的概述。这些内置数据类型是 Python 程序的基础，从而提供了强大而灵活的方法来存储、作数据并处理数据。

<code>bytearray</code> 一个可变的字节数组。
<code>bytes</code> 不可变的字节序列。
<code>complex</code> 复数。
<code>dict</code> 键值对的可变集合。
<code>float</code> 浮点数。
<code>frozenset</code> 唯一且可哈希对象的不可变集合。
<code>set</code> 唯一且可哈希对象的可变集合。
<code>str</code> 不可变的字符序列。
<code>tuple</code> 一个不可变的对象序列。

参考

使用 Loop for

最常见的循环类型是 **for 循环**。您可以使用循环通过三个步骤创建元素列表：for

1. 实例化一个空列表。
2. 遍历一个可迭代对象或一系列元素。
3. 将每个元素附加到列表的末尾。

如果要创建包含前 10 个完美方块的列表，则可以在三行代码中完成这些步骤：

```
蟒
>>> squares = []
>>> for number in range(10):
>>>     get_price_with_tax(price):
...     return price * (1 + TAX_RATE)
...

>>> final_prices = map(get_price_with_tax, prices)
>>> final_prices
<map object at 0x7f34da341f90>

>>> squares = [number * number for number in range(10)]
>>> squares
...     return letter.isalpha() and letter.lower() not in vowels
...

蟒

>>> original_prices = [1.25, -9.45, 10.22, 3.78, -5.92, 1.16]
>>> [price if price > 0 else 0 for price in original_prices]
[1.25, 0, 10.22, 3.78, 0, 1.16]
```

Here, your expression is a conditional expression, . This tells Python to output the value of if the number is positive, but to use 0 if the number is negative. If this seems overwhelming, comprehension.get_price()

The walrus operator (:) solves this problem. It allows you to run an expression while simultaneously assigning the output value to a variable. The following example shows how this is possible, using to generate fake weather data::=get_weather_data()

```
Python
>>> import random
>>> def get_weather_data():
...     [1, 1, 1],
...     [2, 2, 2],
... ]
>>> flat = []
>>> for row in matrix:
...     for number in row:
...         flat.append(number)
...

UNIQUE SWAG FOR PYTHONISTAS
www.nerdlettering.com

🛒 移除广告
33333333283333333333500000000
```

It's up to you whether you prefer the generator expression or .map()

用于优化性能的分析

So, which approach is faster? Should you use list comprehensions or one of their

```
...     prices = []
...     for price in PRICES:
...         prices.append(get_price(price))
...     return prices
...

>>> timeit.timeit(get_prices_with_map, number=100)
2.05543370979998566
```

速习使 且能 列表推导式或到了很多方法。但你的速度和在需要的时候使用数

什么时候应该在 Python 中使用列表推导式而不是循环？

显示/隐藏

如何在 Python 中为列表推导式添加条件逻辑？


显示/隐藏

在 Python 中，列表推导比 for 循环快吗？

显示/隐藏

如何在 Python 中使用列表推导式优化性能？

显示/隐藏

 **立即观看** 本教程包含由 RealPython 团队创建的相关视频课程。与书面教程一起观看以加深您的理解：[了解 Python 列表推导式](#)

Python 技巧

每隔几天就会收到一个简短而甜蜜的 Python 技巧发送到您的收件箱。从来没有垃圾邮件。随时取消订阅。由 RealPython 团队策

```
1 # How to merge two dicts
2 # in Python 3.5+
3
4 >>> x = {'a': 1, 'b': 2}
5 >>> y = {'b': 2, 'c': 4}
10 {'c': 4, 'a': 1, 'b': 3}
```

电子邮件地址

向我发送 Python 技巧 »

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

```
(base) 14332@cnnyj:~/repo$ ls -l
total 12
-rwxr-xr-x 1 14332 197609 0 3月 19 21:41 myproject/
-rwxr-xr-x 1 14332 197609 0 3月 19 18:38 prj1/
-rwxr-xr-x 1 14332 197609 0 3月 18 22:51 week02/
-rwxr-xr-x 1 14332 197609 0 3月 29 11:45 week04/
-rwxr-xr-x 1 14332 197609 0 3月 29 13:06 week04_9383/
-rwxr-xr-x 1 14332 197609 0 4月 7 18:38 week05/
(base) 14332@cnnyj:~/repo$ cat week04_9383/environment.yml
name: week04_9383
channels:
  - conda-forge
dependencies:
  - python3.12
(base) 14332@cnnyj:~/repo$ cp week04_9383/environment.yml week05/
(base) 14332@cnnyj:~/repo$ ls -l
total 16
-rwxr-xr-x 1 14332 197609 0 3月 19 21:41 myproject/
-rwxr-xr-x 1 14332 197609 0 3月 19 18:38 prj1/
-rwxr-xr-x 1 14332 197609 0 3月 18 22:51 week02/
-rwxr-xr-x 1 14332 197609 0 3月 29 11:45 week04/
-rwxr-xr-x 1 14332 197609 0 3月 29 13:06 week04_9383/
-rwxr-xr-x 1 14332 197609 0 4月 7 18:32 week05/
(base) 14332@cnnyj:~/repo$ cat week05/environment.yml
name: week04_9383
channels:
  - conda-forge
dependencies:
  - python3.12
  - wat-inspector(base) 14332@cnnyj:~/repo$
```

```
(base) 14332@cnnyj:~/repo/week05 (main)$ conda env create
D:\anzhuang\an\Lib\argparse.py:2006: FutureWarning: 'remote_definition' is deprecated 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/pkg/main
  - https://repo.anaconda.com/pkg/r
  - https://repo.anaconda.com/pkg/nsys2
Platform: win-64
Collecting package metadata (repodata.json): /
```

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

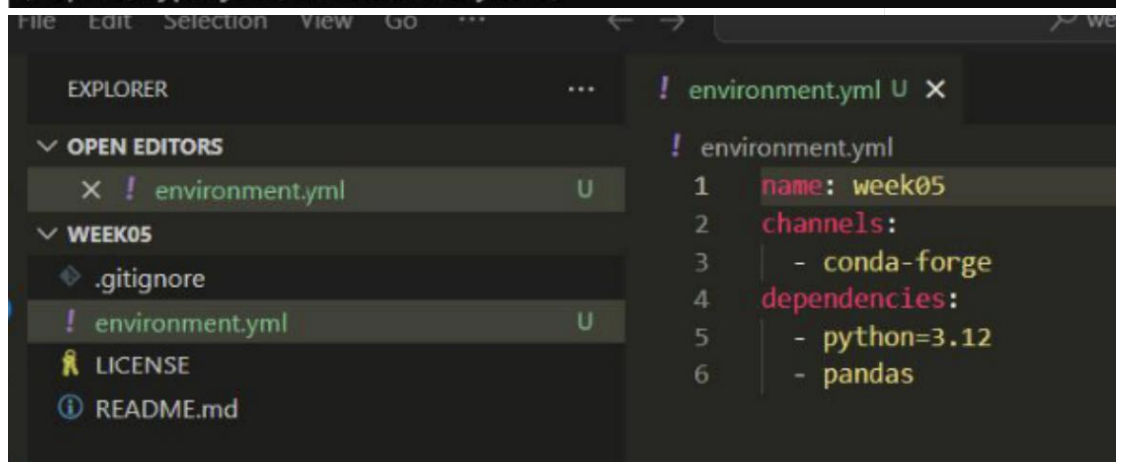
```
(base) Administrator@PC-20220115W0JX MINGW64 ~/repo
$ cd week05

(base) Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ ls -l ../myproject
total 52114
-rw-r--r-- 1 Administrator 197121      87  3月 24 01:28 environment.yml
-rw-r--r-- 1 Administrator 197121 53362688  3月 24 01:53 EPA_SmartLocation.py
-rw-r--r-- 1 Administrator 197121     713  3月 24 01:52 main.py

(base) Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ cat ../myproject/environment.yml
name: myproject
channels:
  - conda-forge
dependencies:
  - python=3.12
  - pandas

(base) Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ cp ../myproject/environment.yml
cp: missing destination file operand after '../myproject/environment.yml'
Try 'cp --help' for more information.

(base) Administrator@PC-20220115W0JX MINGW64 ~/repo/week05 (main)
$ cp ../myproject/environment.yml ./
```



2 和 3 参考了仓库内其他同学的学习报告

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)

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

1. **pdb 调试试验**：在 Python 调试器中逐行测试代码，观察对象行为。

python



```
import pdb; pdb.set_trace() # 插入断点，运行后用 `expr` 命令测试表达式
```

2. **代码文件编写**：将验证通过的代码写入 `use_of_{name}.py`（如 `use_of_list.py`），示例结构：

python



```
# use_of_list.py
# 字面值创建
lst = [1, 2, 3]
# 推导式
even_lst = [x for x in range(10)
            if x % 2 == 0]
# 验证运算符
print(lst + [4, 5]) # 支持 + 拼接
print(len(lst))    # 支持 len()
```

通过以上步骤，可系  握每种对象类型的使用方式和特性。

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

注意事项

1. 本周讲到许多抽象概念，比如“表达式”、“对象”、“类型”、“实例”、“字面值”、“初始化”、“提取值”、“可迭代”等等。很多同学会觉得抽象概念难以理解，我们的建议是通过 **例子** 来理解和把握 **抽象**，这个道理和学习 **数学** 是一样的。录播里会讲到各种具体类型，都是这些抽象概念的例子。也有很多同学会觉得抽象概念没什么用，这其实是非常大的误解。如果只学习各种具体类型，不把握抽象概念，就会深入茫茫大海中不能浅出，不能举一反三、触类旁通，会学得很痛苦。这个道理也和学习 **数学** 是一样的，抽象数学才有泛化 (**generalize**) 能力，能够让洞见在学科之间迁移。录播尝试通过具体例子来讲解抽象概念，一旦同学们能够理解和把握住 **抽象**，那么将来遇到任何新的具体类型，都将能够轻松掌握。
2. Python 语言的概念明显要比 Stata、SAS、R、Matlab 等语言多。后者只是特定领域 (**domain-specific**) (比如计量、统计、数值计算) 的语言，而前者则是一般意义 (**general-purpose**) 上的编程语言。现在人们使用计算机已经越来越不满足于被限定在特定领域，而是需要将来自多种不同系统、平台或服务的数据进行 **集成**。未来人们的工作也很有可能转向为：基于自己的专有知识或数据，开发提供“微服务” (**micro-service**)，融入世界范围的“云计算” (**cloud-computing**) 体系，收取服务费 (软件即服务，**Software as a Service, SaaS**) 作为劳动和智力的报酬。金融服务未来会 (还是已经?) 演变为云服务模式，所以金融学子只掌握统计语言 (数据分析，**Data Analysis**) 是不够的，必须学习更多的、更一般的编程概念，掌握数据工程 (**Data Engineering**) 的能力。
3. 计算机的虚拟世界远比现实世界更为丰富多彩和迷人。现实世界只有一个，虚拟世界却可以有无数个；现实世界受到各种客观规律的制约，虚拟世界里我们自己扮演“造物主”，自己创造概念；虚拟世界在开源协作的浪潮下，更是不断加速地在推陈出新。我们认为，掌握 Python 的“对象”思维，可能是进入这个虚拟世界最佳的入场券。