

STAT243 Lecture 5.4 Types and Data Structures

1 Types 和 Classes

Logic ▾

Type 指定了

- 给定信息如何存储
- 可以对信息执行哪些操作

"Primitive" types 是最基本的类型, 通常直接关系到数据在内存或磁盘上的存储方式 (例如 boolean, integer, numeric, character, pointer 等等)

1.1 Static vs. Dynamic Typing

Static 和 Dynamic Typing 是什么?

- 在像 C 和 C++ 这样的编译语言中, 必须定义每个变量的类型. 因此这些语言是 **statically typed**
- 像 Python 和 R 这样的解释型语言是 **dynamically typed**. 可以在不同时间将不同 type 的信息与给定的变量名关联, 而无需声明变量的 type



Python

```
1 x = 'hello'
2 print(x) # hello
3 x * 3 # 'hellohellohello'
4
5 x = 7
6 x * 3 # 21
7
8 x = {'mykey': 7}
9 x * 3 # TypeError: unsupported operand type(s) for *: 'dict' and 'int'
```

Static 和 Dynamic Typing 的优缺点:

- Dynamic typing 有助于快速实现, 可以
 - 避免繁琐的类型定义
 - 避免类型间的微小不一致所导致的问题 (e.g. 整数乘以实数)
- Static typing 有助于软件开发, 可以
 - 防止因 mismatched values 和 unexpected user inputs 所导致的错误
 - 通常更快的执行速度 (因为在运行代码时不需要检查变量类型)

⚠ Remark ▾

Python (和 R) 中更复杂的类型通常使用 references (pointers). 我们将在讨论 memory 时详细研究

1.2 Python 中的类型

Logic ▾

Python 中重要的内置数据类型包括列表, 元组和字典, 以及基本标量类型如整数, 浮点数和字符串

查看 Python 和 numpy 中各种内置数据结构的类型.

```
Python
1 x = 3
2 type(x) # <class 'int'>
3
4 x = 3.0
5 type(x) # <class 'float'>
6
7 x = 'abc'
8 type(x) # <class 'str'>
9
10 x = False
11 type(x) # <class 'bool'>
12
13 x = [3, 3.0, 'abc']
14 type(x) # <class 'list'>
15
16 import numpy as np
17 x = np.array([3, 5, 7]) # 整数数组
18 type(x) # <class 'numpy.ndarray'>
19 type(x[0]) # <class 'numpy.int64'>
20
21 x = np.random.normal(size=3) # 浮点数数组
22 type(x[0]) # <class 'numpy.float64'>
23
24 x = np.random.normal(size=(3, 4)) # 多维数组
25 type(x) # <class 'numpy.ndarray'>
```

有时 numpy 可能会修改 data types, 这通常效果很好, 但有时可能需要我们自己控制 types.

```
Python
1 x = np.array([3, 5, 7.3])
2 x # array([3., 5., 7.3])
3 type(x[0]) # <class 'numpy.float64'>
4
5 x = np.array([3.0, 5.0, 7.0]) # 强制使用浮点数
6 type(x[0]) # <class 'numpy.float64'>
7
8 x = np.array([3, 5, 7], dtype='float64')
9 type(x[0]) # <class 'numpy.float64'>
```

⚠ Remark ▾

在使用 GPU 时会出现这种情况, 默认通常使用 32 位数字而不是 64 位数字.

1.3 Composite objects

许多对象可以是 **composite** 的 (例如字典列表或包含列表, 元组和字符串的字典).

```
Python
1 mydict = {'a': 3, 'b': 7}
2 mylist = [3, 5, 7]
3 mylist[1] = mydict
4 mylist # [3, {'a': 3, 'b': 7}, 7]
5 mydict['a'] = mylist
```

1.4 Mutable objects 可变对象

Python 中的大多数对象可以就地修改 (即仅修改对象的某些部分), 但 tuple, strings 和 sets 是 **immutable** 的。

```
Python
1 x = (3, 5, 7)
2 try:
3     x[1] = 4
4 except Exception as error:
5     print(error)
6 # 'tuple' object does not support item assignment
7
8 s = 'abc'
9 s[1]
10 # 'b'
11
12 try:
13     s[1] = 'y'
14 except Exception as error:
15     print(error)
16 # 'str' object does not support item assignment
```

1.5 Converting between types

我们可以在不同的 basic types 之间进行 **cast (coerce)**

⚠ Remark ▾

在 compiled languages 中通常需要显式进行强制转换, 而在像 Python 这样的 interpreted languages 中则较少需要

```
Python
1 y = str(x[0])
2 y # '3'
3
4 y = int(x[0])
5 type(y) # <class 'int'>
```

一些常见的转换包括:


- 将被解释为字符串的数字转换为实际数字
- 在布尔值和数值之间进行转换

在某些情况下, Python 会以智能的方式 (或偶尔不那么智能的方式) 在后台自动进行转换. 考虑这些隐式强制转换的示例.

```
Python
1 x = np.array([False, True, True])
2 x.sum() # 2
3
4 x = np.random.normal(size=5)
5 try:
6     x[3] = 'hat' # ValueError
7 except Exception as error:
8     print(error) # could not convert string to float: 'hat'
9
10 myList = [1, 3, 5, 9, 4, 7]
11
12 myList[2.0]
13 # TypeError: list indices must be integers or slices, not float
14 myList[2.73]
```

```
15 # TypeError: list indices must be integers or slices, not float
```

R 的严格性较低, 会在某些 Python 不会进行转换的情况下进行转换.

 R

```
1 x <- rnorm(5)
2 x[2.0]
3 # [1] -0.4702201
4
5 x[2.73]
6 # [1] -0.4702201
```

1.6 Python Object Protocols

有多种 **broad categories** of kinds of objects:

- mapping
- number
- sequence
- iterator

这些称为 **object protocols**, 属于给定类别的所有对象共享关键特征

Example ▾

- sequence 对象具有长度和通过索引访问 (有序) 元素的概念
- 迭代器对象具有 "下一个" 和 "停止" 的概念.

⚠ Remark ▾

Object protocols 基于 Python object model 的 **dunder (双下划线)** 方法实现