Python编程及人工智能应用

第二章 Python语言基础

https://bolei-zhang.github.io/course/python-ai.html

内容提要

- Python基本语法
  - 变量和数据类型
  - 运算符和表达式
  - 字符串
  - 流程控制
- Python组合数据类型
- Python函数
- 异常处理和文件操作
- Python面向对象基础
- 数值计算库NumPy

交互模式vs脚本模式

- 交互模式
  - 直接在命令行输入python或python3，每次回车之后运行一行代码

- 脚本模式
  - 编辑后缀为.py的文件
  - 然后通过 python [文件名].py运行

  - Windows推荐用windows terminal
  - MAC推荐用Iterm
  - 自学一些简单的Linux shell命令
    - ls, pwd, cd, which, ps…

Python设计

- 数据类型
  - Python的变量不用声明，对象是储存在内存中的实体。但我们并不能直接接触到该对象。我们在程序中写的对象名，只是指向这一对象的引用(reference)。
  - Python对象的类型和内存都是在运行时确定的
- 编译
  - 当Python程序运行时，编译的结果则是保存在位于内存中的PyCodeObject中，当Python程序运行结束时，Python解释器则将PyCodeObject写回到pyc文件中。
  - 当Python程序第二次运行时，首先程序会在硬盘中寻找对应的pyc文件，如果找到，则直接载入，否则就重复上面的过程。
- 指针
  - Python函数的参数是引用传递（除非对于不可变类型）
- 内存管理
  - Python采用了类似Windows内核对象一样的方式来对内存进行管理。每一个对象，都维护这一个对指向该对对象的引用的计数
  - 当内存中有不再使用的部分时，垃圾收集器就会把他们清理掉。它会去检查那些引用计数为0的对象，然后清除其在内存的空间。

推荐资料：https://docs.python.org/zh-cn/3.7/faq/design.html

Python语法基本特点

- 使用缩进来表示程序的层次结构。不同于C语言，Python变量可不声明直接使用
  - 同一层次代码块，缩进所包含空格数量必须一致
    - #代码2.1 接收用户从键盘上输入的整数，并判别其奇偶性
    - num = int(input(“请您输入一个整数：”))
    - if num%2==0:
    - print(num,"是一个偶数")
    - print("这里执行的是程序中的第1个分支")
    - else:
    - print(num,"是一个奇数")
    - print("这里执行的是程序中的第2个分支")
- 输入函数input()，从键盘输入一个字符串
  - int()函数用于把字符串转换成整数
- 输出函数print()，输出后自动换行

为什么Python使用缩进

Guido van Rossum 认为使用缩进进行分组非常优雅，并且大大提高了普通Python程序的清晰度。大多数人在一段时间后就学会并喜欢上这个功能。

由于没有开始/结束括号，因此解析器感知的分组与人类读者之间不会存在分歧。偶尔C程序员会遇到像这样的代码片段:

```
if (x <= y)
        x++;
        y--;
z++;
```

如果条件为真，则只执行 `x++` 语句，但缩进会使你认为情况并非如此。即使是经验丰富的C程序员有时会长时间盯着它，想知道为什么即使 `x > y` ， `y` 也在减少。

因为没有开始/结束括号，所以Python不太容易发生编码式冲突。在C中，括号可以放到许多不同的位置。如果您习惯于阅读和编写使用一种风格的代码，那么在阅读（或被要求编写）另一种风格时，您至少会感到有些不安。

许多编码风格将开始/结束括号单独放在一行上。这使得程序相当长，浪费了宝贵的屏幕空间，使得更难以对程序进行全面的了解。理想情况下，函数应该适合一个屏幕（例如，20--30行）。 20行Python可以完成比20行C更多的工作。这不仅仅是由于缺少开始/结束括号 -- 缺少声明和高级数据类型也是其中的原因 -- 但缩进基于语法肯定有帮助。

Python对象及类型

- Python 所有数据都由对象或对象间关系来表示
- Python常见的内置标准类型
  - 整型数字（int）
  - 浮点型数字（float）
  - 逻辑值（bool）
  - 字符串（str）
  - 复数型数字（complex）
  - 列表（list）、元组（tuple）、可变集合（set）、不可变集合（frozenset）、字典（dict）

Python对象及类型

```
#代码2.2 在屏幕上输出各种对象的类型
print("100 is", type(100))
print("3.14 is", type(3.14))
print("5+2j is", type(5+2j))
print("True and False are", type(False))
print("\'I love Python.\' is", type("I love Python."))
print("[1,2,3] is", type([1,2,3]))
print("(1,2,3) is", type((1,2,3)))
print("{1,2,3} is", type({1,2,3}))
print("frozenset({1,2,3}) is",
      type(frozenset({1,2,3})))
print("{'name':'Tom','age':18} is",
      type({'name':'Tom','age':18}))
```

代码2.2运行后输出如下：

```
100 is <class 'int'>
3.14 is <class 'float'>
5+2j is <class 'complex'>
True and False are <class 'bool'>
'I love Python.' is <class 'str'>
[1,2,3] is <class 'list'>
(1,2,3) is <class 'tuple'>
{1,2,3} is <class 'set'>
frozenset({1,2,3}) is <class 'frozenset'>
{'name':'Tom','age':18} is <class 'dict'>
```

- type()是Python的内置函数，返回对象所属类型

Python变量和关键字

- 标识符的命名规则
  - 由大写和小写字母、下划线 _ 以及数字组成
  - 不可以数字打头，可以包含Unicode字符（中文）
  - 不可以和Python中的关键字重复
- Python关键字

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Python变量赋值

- 赋值语句可以将变量和对象进行关联

```
#代码2.3 使用赋值语句建立变量和对象之间的关联
#这是使用科学记数法表示浮点数0.0178的方法
var = 1.78E-2
print(var, type(var))
#复数的表示中，带有后缀j或者J的部分为虚部
var = 3+7j
print(var, type(var))
#字符串数据既可以用单引号，也可以用双引号进行定义
var = "人生苦短，我用Python"
print(var, type(var))
```

```
运行代码2.3输出如下
0.0178 <class 'float'>
(3+7j) <class 'complex'>
人生苦短，我用Python <class 'str'>
```

```
#代码2.4 赋值语句的连续赋值和多重赋值
#用于将不同的变量关联至同一个对象
x = y = z = 100
print(x, y, z)
#用于将不同的变量关联至不同的对象
a, b, c = 7, 8, 9
print(a, b, c)
```

```
运行代码2.4后输出如下
交换之前x与y的值为： 3 5
交换之后x与y的值为： 5 3
```

Python运算符和表达式

- 表达式可以由多个运算对象和运算符构成
- Python按照运算符的优先级求解表达式的值
  - Python常见运算符（从最低优先级到最高优先级）；Python没有三目运算符，但是可以用if else

运算符	运算符描述
if -- else	条件运算符
or	逻辑或运算
and	逻辑与运算
not x	逻辑非运算
in、not in、is、is not、<、<=、>、>=、!=、==	比较运算
|	按位或运算
^	按位异或运算
&	按位与运算
<<、>>	移位运算
+、-	算术运算符：加、减
*、/、//、%	算术运算符：乘、除、整除、取模（求余数）
+x、-x、~x	单操作数运算符：正、负、按位非运算
**	乘方运算符

Python运算符和表达式

```
#代码2.6 表达式举例
print(1+2*3-4)          #由于乘号的优先级高于加号和减号，所以2*3会优先计算
print(2*3**3)           #乘方运算的优先级比乘法运算还要高
a, b = 10, 20
print(a if a>b else b)  #输出变量a和b中数值较大的那一个
print(True+1)           #运算过程中，True和False可以被自动转换成1和0使用
print(3>2>1)            #连续的比较运算，其结果与表达式3>2 and 2>1等价
print((3>2)>1)          #这条表达式计算的是子表达式3>2的值是否大于1，即True>1
```

运行代码2.6后输出如下：

```
3
54
20
2
True
False
```

Python字符串

- 字符串类型的对象使用一对单引号、一对双引号或者一对三引号进行定义
  - 使用三引号定义字符串的时候可以包含换行符

```
s = '''Hello
    World'''
```

- Python字符串转义字符

转义字符	含义	转义字符	含义
\ '	单引号	\t	水平制表符
\"	双引号	\v	垂直制表符
\\	字符“\”本身	\r	回车符
\a	响铃	\f	换页符
\b	退格符	\ooo	以最多3位的八进制数作为编码值对应的字符
\n	换行符	\xhh	以必须为2位的十六进制数作为编码值对应的字符

Python字符串举例

```
#代码2.7 定义字符串的几种方法
str1 = '单引号可以用来定义字符串'
str2 = "双引号也可以用来定义字符串"
print(str1, str2)
str3 = '''三引号定义的字符串可以直接换行
这是字符串的第二行
这是字符串的第三行'''
print(str3)
print("转义字符\n表示换行符")
print(r"转义字符\n表示换行符")
#格式标记.2f表示保留小数点后2位小数
print(f"半径为5的圆的面积是{3.14*5**2:.2f}")
```

运行代码2.7后输出如下：

```
单引号可以用来定义字符串 双引号也可以用
来定义字符串
三引号定义的字符串可以直接换行
这是字符串的第二行
这是字符串的第三行
转义字符
表示换行符
转义字符\n表示换行符
半径为5的圆的面积是78.50
```

Python字符串是不可变的

Python字符串运算

- f-string，亦称为格式化字符串常量，从Python3.6新引入，目的是使格式化字符串操作更加简便
- f-string在形式上是以f或F修饰符引领的字符串
  - 格式：f”{表达式}”

```
#代码2.8 与字符串有关的运算
#字符串之间用加法运算连接，结果为连接后的字符串
print("ABCD"+"1234")
#字符串和整数进行乘法运算，结果为字符串复制多遍并连接
print("Hi"*3)
str1 = "I love Python!"
#获取字符串str1中的首个字符
print(f"str1[0] = {str1[0]}")
#获取字符串str1的最后一个字符
print(f" str1[-1] = {str1[-1]}")
#获取字符串中索引号从2开始到6之前结束的子字符串，即"love"
print(f" str1[2:6] = {str1[2:6]}")
```

运行代码2.8后输出如下：

```
ABCD1234
HiHiHi
str1[0] = 'I'
str1[-1] = '!'
str1[2:6] = 'love
```

Python字符串的常用函数

- 字符串对象有一系列已定义好的函数可直接使用

```
#代码2.9 字符串对象的常用方法
str1 = "i have an apple."
print(f" str1.capitalize() = {str1.capitalize()}") #用于将字符串的首字母大写
print(f" str1.title() = {str1.title()}") #用于将字符串中每个单词的首字母大写
print(f" str1.count('a') = {str1.count('a') }") #用于统计指定的字符在字符串中出现的次数
print(f" str1.startswith('i am') = {str1.startswith('i am') }") #用于判定字符串是否以指定的参数开始
print(f" str1.endswith('apple.') = {str1.endswith('apple.') }") #用于判定字符串是否以指定的参数结尾
print(f" str1.find('apple') = {str1.find('apple') }") #用于查找指定的子字符串并返回其起始位置
print(f" str1.find('pear') = {str1.find('pear') }") #若无法找到指定的子字符串，find()方法会返回-1
print(f " str1.split() ={str1.split() }" ) #对字符串进行切割，默认为所有的空字符，包括空格、换行(\n)、制表符(\t)等。作为分隔符
print(f"','.join(['a','b','c']) = {','.join(['a','b','c']) }") #用指定的字符串将若干字符串类型的对象进行连接
print(f" 'ABcd'.upper() = {'ABcd'.upper() }") #将字符串中的字母转换成大写
print(f" 'ABcd'.lower() = {'ABcd'.lower() }") #将字符串中的字母转换成小写
print(f" '  ABcd  '.strip() = {'  ABcd  '.strip() }") #删除字符串前后的特殊字符，比如空格和换行符
```

运行代码2.9后的输出结果：

```
str1.capitalize() = 'I have an apple.'
str1.title() = 'I Have An Apple.'
str1.count('a') = 3
str1.startswith('i am') = False
str1.endswith('apple.') = True
str1.find('apple') = 10
str1.find('pear') = -1
str1.split() = ['i', 'have', 'an', 'apple.']
','.join(['a','b','c']) = 'a,b,c'
'ABcd'.upper() = 'ABCD'
'ABcd'.lower() = 'abcd'
'  ABcd  '.strip() = 'ABcd'
```

Python选择结构

- 用if...elif...else来构造选择结构的程序代码
- python没有switch语句

```
#代码2.10 将百分制成绩转换成五级制成绩
score = float(input("请输入一个百分制成绩："))
if score>=90:
    print(f"成绩{score}对应的五级制成绩是优秀")
elif score>=80:
    print(f"成绩{score}对应的五级制成绩是良好")
elif score>=70:
    print(f"成绩{score}对应的五级制成绩是中等")
elif score>=60:
    print(f"成绩{score}对应的五级制成绩是及格")
else:
    print(f"成绩{score}对应的五级制成绩是不及格")
```

运行代码2.10后的输出结果：

```
请输入一个百分制成绩：95
成绩95.0对应的五级制成绩是优秀
```

为什么Python中没有switch或case语句?

你可以通过一系列 `if... elif... elif... else`.轻松完成这项工作。对于switch语句语法已经有了一些建议，但尚未就是否以及如何进行范围测试达成共识。有关完整的详细信息和当前状态，请参阅 **PEP 275** 。

对于需要从大量可能性中进行选择的情况，可以创建一个字典，将case 值映射到要调用的函数。例如：

```
def function_1(...):
    ...

functions = {'a': function_1,
             'b': function_2,
             'c': self.method_1, ...}

func = functions[value]
func()
```

Python选择结构

- 通过组合多个if...elif...else结构进行嵌套使用

```
#代码2.11 判断从键盘上输入的三个整数构成的三角形形状
a,b,c = input("请输入组成三条边，用空格分隔：").split()
a,b,c = int(a), int(b), int(c)
a,c,b = min(a,b,c), max(a,b,c), a+b+c-min(a,b,c)-max(a,c,b)
if a+b<=c:
    print("输入的整数无法构成三角形")
else:
    if a==b==c:
        print("输入的整数构成一个等边三角形")
    elif a==b:
        print("输入的整数构成一个等腰三角形")
    elif a**2+b**2==c**2:
        print("输入的整数构成一个直角三角形")
    else:
        print("输入的整数构成一个普通三角形")
```

运行代码2.11后的输出结果：

```
请输入组成三角形三条边的整数，用空格分隔：1 1 2
输入的整数无法构成三角形
请输入组成三角形三条边的整数，用空格分隔：3 3 3
输入的整数构成一个等边三角形
请输入组成三角形三条边的整数，用空格分隔：3 3 5
输入的整数构成一个等腰三角形
请输入组成三角形三条边的整数，用空格分隔：3 4 5
输入的整数构成一个直角三角形
请输入组成三角形三条边的整数，用空格分隔：5 6 7
```

Python不能在表达式中赋值

Python循环结构

- 用关键字while或者for可以用来构造循环结构
  - while语句实现累加
  - for语句实现累加

```
#代码2.12  使用while循环完成1至100所有整数求和
n = 1
s = 0
while n<=100:
    s += n          #与s = s + n等价
    n += 1          #与n = n + 1等价
print("1到100所有整数的和是：",s)
```

```
#代码2.13 使用for循环完成1至100所有整数求和
s = 0
for n in range(1,101):
    s += n
print("1到100所有整数的和是：",s)
```

- range(1,101)所表示的从1到101以内（不含101）的所有整数

Python循环结构

- 如果range()函数只有一个参数，则该参数表示返回结果的终止值（不含）
- 如果range()函数有两个参数，则第1个参数表示返回结果的起始值，第2个参数表示返回结果的终止值（不含）
- 如果range()函数有三个参数，则第1个参数表示返回结果的起始值，第2个参数表示返回结果的终止值（不含），第3个参数表示每次步进的增量值

```
#代码2.14 求100以内所有奇数的和
#sum()函数的功能为对其参数进行求和
s = sum(range(1,100,2))
print("100以内所有奇数的和是：",s)
```

Python循环结构

- break语句会终结最近的外层循环，如果循环有可选的else子句，也会跳过该子句
- continue语句会终止当前循环中剩下的语句，并继续执行最近的外层循环的下一个轮次

```
#代码2.15 输出从小到大排列的第100个素数
count = 0
num = 2
while True:
    #下方的for循环用于判断num是否为素数，如果是则计数器count加1
    for i in range(2,num):
        if num%i==0:
            break
    else:
        count += 1
```

```
#如果计数器小于100，num加1，且终止当前循环中剩下的语句，并继续执行循环的下一个轮次
    if count<100:
        num += 1
        continue
    #剩下的代码只有在上方if不成立的时候才会被执行到，即计数器count为100的情况下
    print(num,"是从小到大排列的第100个素数")
    break                    #输出完毕后，使用break语句终结外层的while循环
```

一起读代码

```python
"""
A pure Python implementation of the quick sort algorithm

For doctests run following command:
python3 -m doctest -v quick_sort.py

For manual testing run:
python3 quick_sort.py
"""
from __future__ import annotations


def quick_sort(collection: list) -> list:
    """A pure Python implementation of quick sort algorithm

    :param collection: a mutable collection of comparable items
    :return: the same collection ordered by ascending

    Examples:
    >>> quick_sort([0, 5, 3, 2, 2])
    [0, 2, 2, 3, 5]
    >>> quick_sort([])
    []
    >>> quick_sort([-2, 5, 0, -45])
    [-45, -2, 0, 5]
    """
    if len(collection) < 2:
        return collection
    pivot = collection.pop()  # Use the last element as the first pivot
    greater: list[int] = []  # All elements greater than pivot
    lesser: list[int] = []  # All elements less than or equal to pivot
    for element in collection:
        (greater if element > pivot else lesser).append(element)
    return quick_sort(lesser) + [pivot] + quick_sort(greater)


if __name__ == "__main__":
    user_input = input("Enter numbers separated by a comma:\n").strip()
    unsorted = [int(item) for item in user_input.split(",")]
    print(quick_sort(unsorted))
```

https://github.com/TheAlgorithms/Python/blob/master/sorts/quick_sort.py